PS 390 DOCUMENT SET

REFERENCE MATERIALS 1–4

The contents of this document are not to be reproduced or copied in whole or in part without the prior written permission of Evans & Sutherland. Evans & Sutherland assumes no responsibility for errors or inaccuracies in this document. It contains the most complete and accurate information available at the time of publication, and is subject to change without notice.

PS 300, PS 330, PS 340, PS 350, PS 390, and Shadowfax are trademarks of the Evans & Sutherland Computer Corporation.

Copyright © 1987
EVANS & SUTHERLAND COMPUTER CORPORATION
P.O. Box 8700, 580 Arapeen Drive
Salt Lake City, Utah 84108

REFERENCE MATERIALS

The *Reference Materials RM1–4 and RM5–16* provide reference information for the user of the PS 390 system. Summaries of the ASCII commands, intrinsic functions, initial function instances and GSRs are contained in the first part of the volume. Included in the second part of the volume are sections covering interactive devices, interfaces and options, host input data flow, system function network diagrams, diagnostic utilities, system errors and host communications. The final section contains an index to the complete *PS 390 Document Set.*

RM1 Command Summary

This section contains a summary of the ASCII form of each PS 390 command. The long form and acceptable short form of each command are given, together with information on parameters, default values, and other requirements. Where a command creates a node in a display structure, the type of node is indicated. If that node can be updated with values from an interactive device, the inputs to the node and acceptable data types are shown in the diagram. Examples of the use of commands are given when appropriate, and related information is included as notes. The summary is alphabetized for ease of use. Appendices list commands by classification, give the syntax of each command, and provide a cross-reference to the GSRs found in *Section RM4.*

RM2 Intrinsic Functions

This section contains a summary of information about each PS 390 intrinsic user function and each intrinsic system function. Functions are represented as boxes with numbered inputs and outputs and acceptable data types. Default values, associated functions, notes and examples are listed when appropriate. An appendix lists the functions by classification.

RM3 Initial Function Instances

This section contains a summary of information about PS 390 initial function instances. Initial function instances are represented as boxes with numbered inputs and outputs and acceptable data types. Default values, associated functions, notes and examples are listed when appropriate. An appendix lists the initial function instances by classification.

RM4 Graphics Support Routines

This section contains a summary of the Graphics Support Routines (GSRs). GSRs corresponding to ASCII commands and utility and raster routines are included. Descriptions of the VAX and IBM FORTRAN, VAX and IBM Pascal, and UNIX/C GSRs are listed. An appendix provides a cross-reference to the ASCII commands documented in *Section RM1*.

RM1. COMMAND SUMMARY

CONTENTS

APPLIED TO/THEN 3
ATTRIBUTES 4
BEGIN...END 7
BEGIN_FONT...END_FONT 8
BEGIN_S...END_S 10
BSPLINE 13
CANCEL XFORM 16
CHARACTER FONT 17
CHARACTER ROTATE 18
CHARACTERS 20
CHARACTER SCALE 22
COMMAND STATUS 24
CONFIGURE 25
CONNECT 26
COPY 27
DECREMENT LEVEL_OF_DETAIL 29
DELETE 30
DISCONNECT 31
DISPLAY 32
ERASE PATTERN FROM 33
EYE 34
FIELD_OF_VIEW 36
FINISH CONFIGURATION 38
FOLLOW WITH 39
FORGET (Structures) 41
FORGET (Units) 42
(Function Instancing) 43
GIVE_UP_CPU 44
IF CONDITIONAL_BIT 45
IF LEVEL_OF_DETAIL 47

IF PHASE 49
ILLUMINATION 50
INCLUDE 52
INCREMENT LEVEL_OF_DETAIL 53
INITIALIZE 54
INSTANCE OF 56
LABELS 57
LOAD VIEWPORT 59
LOOK 61
MATRIX_2x2 64
MATRIX_3x3 66
MATRIX_4x3 68
MATRIX_4x4 70
(Naming of Display Structure Nodes) 72
NIL 73
OPTIMIZE MEMORY 74
OPTIMIZE STRUCTURE;...END OPTIMIZE; 75
PATTERN 77
PATTERN WITH 78
POLYGON 79
POLYNOMIAL 82
PREFIX WITH 84
RATIONAL BSPLINE 85
RATIONAL POLYNOMIAL 89
RAWBLOCK 92
REBOOT 94
REMOVE 95
REMOVE FOLLOWER 96
REMOVE FROM 97
REMOVE PREFIX 98
RESERVE_WORKING_STORAGE 99
!RESET 101
ROTATE 102
SCALE 104
SECTIONING_PLANE 106
SELECT FILTER 108
SEND 110
SEND number*mode 111
SEND VL 112
SET BLINKING ON/OFF 113
SET BLINK RATE 114
SET CHARACTERS 115

SET COLOR 116
SET CONDITIONAL_BIT 118
SET CONTRAST 120
SET DEPTH_CLIPPING 122
SET DISPLAYS 124
SET INTENSITY 126
SET LEVEL_OF_DETAIL 128
SET LINE_TEXTURE 130
SET PICKING 132
SET PICKING IDENTIFIER 134
SET PICKING LOCATION 135
SET PRIORITY 137
SET RATE 138
SET RATE EXTERNAL 140
SETUP CNESS 142
SETUP INTERFACE 144
SETUP PASSWORD 145
SHOW INTERFACE 146
SOLID_RENDERING 147
STANDARD FONT 152
STORE 153
SURFACE_RENDERING 154
TEXT SIZE 159
TRANSLATE 161
VARIABLE 163
VECTOR_LIST 164
VIEWPORT 169
WINDOW 172
WITH PATTERN 174
WRITEBACK 176
XFORM 178

Appendix A
PS 390 Commands by Category 180

Appendix B
PS 390 Command Syntax 185

Appendix C
ASCII Commands
and Corresponding GSRs 197

ASCII Character Code Set 205

Section RM1
Command Summary

This section is a PS 390 command language reference for graphics programmers who are familiar with the basic operation of the PS 390. It contains a summary of the PS 390 commands. The commands are ordered alphabetically on a letter-by-letter basis. The following information, where relevant, is given for each command:

- Name
- Category and subcategory
- Syntax
- Description
- Parameters
- Defaults
- Notes
- Display structure node created
- Inputs for updating node
- Notes on inputs
- Associated functions
- Examples

Appendix A contains a summary of the commands grouped into categories. Appendix B contains an alphabetical listing of the command syntax. Appendix C contains a list of the commands and the corresponding GSRs.

This section also contains the following system commands.

The SETUP PASSWORD, CONFIGURE, and FINISH CONFIGURATION commands allow you to enter and exit the configure mode.

E&S reserves the right to change the content of the CONFIG.DAT file and the implementation of the CONFIG.DAT file without prior notice. Use of any named

entities or networks instanced in configure mode that have names identical to any names found in the CONFIG.DAT file will result in unpredictable system behavior. E&S will not use any names that are preceded with the three characters CM_.

The SHOW INTERFACE and SETUP INTERFACE commands are used to show or change the default values on ports 1 through 5 on the PS 390 control unit.

The SET PRIORITY command sets the execution priority of a function.

Since some commands require the ASCII decimal equivalent of characters in their parameters, an ASCII chart with decimal values is included after the appendices.

TYPE

STRUCTURE — Explicit Referencing

FORMAT

```
name := operation_command [APPLied to name1];
name := operation_command [THEN name1];
```

DESCRIPTION

Associates a command to the structure which is to be affected by the command.

PARAMETERS

operation_command — A command that creates an operation node in a display structure.

name1 — Structure that will be affected by the command.

NOTE

APPLied to and THEN are synonyms. The terms are completely interchangeable.

DISPLAY STRUCTURE NODE CREATED

The command node with a pointer to the structure name1.

EXAMPLE

```
A:= ROTate in X 45 THEN B;
B:= VECtor_list n=5 1,1 -1,1 -1,-1 1,-1 1,1;
```

ATTRIBUTES

TYPE

RENDERING — Data Structuring

FORMAT

```
name := ATTRIBUTES attributes [AND attributes];
```

DESCRIPTION

Specifies the various characteristics of polygons used in the creation of shaded renderings. For a detailed explanation of defining and interacting with shaded images, consult Section *GT13 Polygonal Rendering.*

PARAMETERS

attributes — The attributes of a polygon are defined as follows:

```
[COLOR h[,s[,i]]] [DIFFUSE d] [SPECULAR s] [OPAQUE t]
```

where

h — is a real number specifying the hue in degrees around the color wheel. Pure blue is 0 and 360, pure red is 120, and pure green is 240.

s — is a real number specifying saturation. No saturation (gray) is 0 and full saturation (full toned colors) is 1.

i — is a real number specifying intensity. No intensity (black) is 0, full intensity (white) is 1.

d — is a real number from 0 to 1 specifying the proportion of color contributed by diffuse reflection versus that contributed by specular reflection. Increasing d makes the surface more matte. Decreasing d makes it more shiny.

s — is an integer from 0 to 255 which adjusts the concentration of specular highlights. The more metallic an object is, the more concentrated the specular highlights.

t — is a real number from 0 to 1 specifying the transparency of the polygon, with 1 being fully opaque and 0 being fully transparent (invisible).

DEFAULTS

If no color is specified, the default is white (s = 0, i = 1). If saturation and intensity are not specified, they default to 1. If only hue and saturation are specified, intensity defaults to 1. If no diffuse attribute is given, d defaults to .75. If no specular attribute is given, s defaults to 4. If no opaque attribute is given, the default is 1 (fully opaque).

NOTES

1. Polygon-attribute nodes are created in mass memory but are not part of a display structure. The attributes specified in an ATTRIBUTES command are assigned to polygons which include a WITH ATTRIBUTES clause. The attributes specified in a WITH ATTRIBUTES clause of a POLYGON command apply to all subsequent polygons until superseded by another WITH ATTRIBUTES clause. If no WITH ATTRIBUTES option is given for a polygon node, default attributes are assumed. The default attributes are 0,0,1 for COLOR, 0.75 for DIFFUSE, 4 for SPECULAR, and 1 for OPAQUE.

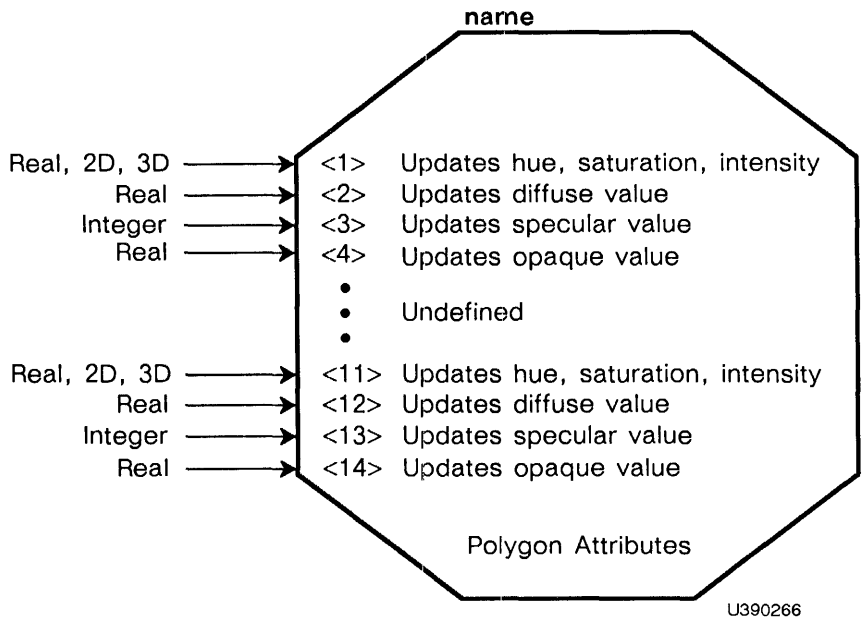
2. The various attributes may be changed from a function network via inputs to an attribute node, but the changes have no effect until a new rendering is created.

3. A second set of attributes may be given after the word AND in the ATTRIBUTES command. These attributes apply to the obverse side of the polygon(s) concerned. In other words, the two sides of an object may have different attributes. The attributes defined in the first attributes pertain to front-facing polygons. Those in the AND attributes clause pertain to back-facing polygons.

DISPLAY STRUCTURE NODE CREATED

Polygon-ATTRIBUTES definition node. This node resides in mass memory, but is not included in a display structure.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

1. Inputs <1> and <11> accept a real number as hue, a 2D vector as hue and saturation, and a 3D vector as hue, saturation and intensity.
2. Values sent to inputs <1>, <2>, and <3> specify the color and attributes for shading the front of the polygon(s) or for both sides if no obverse attributes are given. (Values sent to inputs <11>, <12>, and <13> specify the color and attributes for shading the obverse side of the polygon.)
3. Inputs <4> and <14> accept a real number to update the opaque value of the polygon's attributes.
4. If anything other than a 3D vector is sent to input <1> or <11>, default values for the other variables are assumed.

TYPE

GENERAL — Command Control and Status

FORMAT

```
BEGIN
command;
command;
    .
    .
    .
command;
END;
```

DESCRIPTION

Defines a "batch" of commands which take effect in a single screen update, so that they appear to be executed simultaneously.

PARAMETER

command — Any PS 390 command.

NOTE

Although any command may be used inside a BEGIN...END structure, only commands that create, display, or delete objects will happen "simultaneously."

EXAMPLE

```
BEGIN
DISPlay A;
A:= VECtor_list N=5 1,1 -1,1 -1,-1 1,-1 1,1;
DISPlay B;
B:= VECtor_list N=4 0,0 1,0 1,1 0,0;
END;

{A and B will be displayed simultaneously.}
```

TYPE

MODELING — Character Font

FORMAT

```
name := BEGIN_Font
            [C[0]: N=n {itemized 2D vectors};]
                  .
                  .
                  .
            [C[i]: N=n {itemized 2D vectors};]
                  .
                  .
                  .
            [C[127]: N=n {itemized 2D vectors};]
        END_Font;
```

DESCRIPTION

Defines alternative character fonts, using itemized 2D vector lists to describe each character. Up to 128 PS 390 character codes may be defined for each font.

PARAMETERS

n — Number of vectors in 2D vector list.

i — Decimal ASCII code to be defined. The square brackets around the ASCII number from 0 to 127 are required.

{itemized 2D vectors} — Vectors making up the ASCII character being defined (P x1, y1, L x2, y2, etc.).

NOTES

1. Not all ASCII codes need to be defined for a font. Nothing is output for an undefined character.
2. There is no restriction on the range of values for the 2D vector making up a character, but for correct spacing and orientation to adjacent characters, the range in X and Y should be kept between 0 and 1.

DISPLAY STRUCTURE NODE CREATED

Alternate-character-font definition node. This node resides in mass memory but is not part of a display structure. To specify an alternate font, the character FONT command is used. This creates a character FONT node in a display structure which points to the appropriate alternate font definition.

EXAMPLE

```
A := BEGIN_Font
       C[65]: N=5 P 0,0 L .9,0 L .9,.9 L 0,.9 L 0,0;
     END_Font;
B := BEGIN_Structure
       character FONT A;
       CHARacters 'ABA';
     END_Structure;
DISPlay B;

{Two squares - the new A - will appear right next to each other with
the lower left corner of the first at the origin. The letter B is not
defined in character FONT A, so nothing is DISPlayed for B. Note that
this example creates a special symbol (a square) rather then defining
an alternate character font.}
```

TYPE

STRUCTURE — Implicit Referencing

FORMAT

```
name := BEGIN_Structure
[name1:=] nameable_command;
                .
                .
                .
[namen:=] nameable_command;
        END_Structure;
```

DESCRIPTION

Groups a set of viewing and/or modeling commands so that each element does not need to be explicitly named and APPLied to the next structure in line. This does not, however, prevent naming nested commands directly or explicitly applying a command to another structure via APPLied to.

PARAMETERS

name1..namen — Optional names for individual commands inside the BEGIN_S...END_S, allowing reference to these specific commands from elsewhere (see Note 3). The PS 390 prefixes these names with the name of the outer structure and a period (.). So, for example, the command defined as name1 in the structure is referenced as name.name1.

nameable_command — Nameable commands are those that can be prefixed with "name :=", with the following exceptions:

- COMmand STATus can also be used.
- Intrinsic functions cannot be instanced.
- name := NIL; cannot be used.

NOTES

1. Essentially, any data structuring command except a function instancing command can be used.

2. A non-data command inside a BEGIN_S...END_S is applied to every node that follows in the structure unless it is explicitly APPLied to another structure, in which case it only affects the structure APPLied to (see examples).

3. If a command inside the structure is to be modified later by a function network or from the host, it must be named so that it can be referenced. Its referencing name is the name with all prefixes (e.g., name.name1).

DISPLAY STRUCTURE NODE CREATED

The various nodes created by the nameable commands linked together as specified. The top node of this structure is **name** and is an instance node.

INPUTS FOR UPDATING NODE

The nodes that may be updated are created by those nameable commands that are explicitly named (see note 3). For inputs, refer to the individual command descriptions.

EXAMPLES

```
A:= BEGIN_Structure
    TRANslate by 2,3;
        BEGIN_Structure
            ROTate 30;
            SCALE .5 THEN B;
        END_Structure;
    VECtor_list ... ;
Rot:= ROTate in X 45 THEN C;
    ROTate in Y 90;
    character FONT D THEN E;
Char:= CHARacters 'ABC';
Dat:=  VECtor_list ... ;
    END_Structure;

{To modify the X angle of rotation, a 3x3 matrix would be sent to
<1>A.rot. You could not modify the Y rotation angle since it is not
explicitly named.}
```

```
{An equivalent display structure could be created without using
BEGIN_Structure ... END_Structure, for example:}

A:= INSTance of F;
F:= TRANslate by 2,3 THEN G;
G:= INSTance of H,I,A.Rot,J
H:= INSTance of K;
I:= VECtor_list ...;
A.Rot:= ROTate in X 45 THEN C:
J:= ROTate in X 90 THEN L;
K:= ROTate in Y 30 THEN M;
L:= INSTance of N,A.Char,A.Dat;
M:= SCALE .5 THEN B;
N:= character FONT D THEN E;
A.Char:= CHARacters 'ABC';
A.Dat:= VECtor_list ... ;
```

TYPE

MODELING — Primitives

FORMAT

```
name := BSpline ORDER= k
        [OPEN/CLOSED] [NONPERIodic/PERIodic] [N= n]
        [VERTICES =] x1,y1,[z1]
                     x2,y2,[z2]
                      . . .
                      . . .
                      . . .
                      xn,yn,[zn]
        [KNOTS = t1,t2,...,tj]
        CHORDS = q;
```

DESCRIPTION

Evaluates a B-spline curve, allowing the parametric description of the curve form without the need to specify or transfer the coordinates of each constituent vector.

The B-spline curve C is defined as:

$$C(t) = \sum_{i=1}^{n} piNi,k(t)$$

where

pi = ith vertex of the defining polygon of the B-spline

and

Ni,k = ith B-spline blending function of order k.

The parameter t of the curve and blending functions is defined over a sequence of knot intervals t1,t2,...,tn+k. Different knot sequences define different types of B-splines.

Two common knot sequences are uniform nonperiodic and uniform periodic. A uniform nonperiodic B-spline is defined by the knot sequence:

```
      0      (for j < k)
tj = j-k     (for k < j < n)
     n-k+1   (for n < j < n+k)
```

A uniform periodic B-spline is defined by the knot sequence:

```
tj = j (for j < n+k)
```

The blending functions can be defined recursively as

```
Ni,1(t) = 1 (if ti < t < ti+1), 0 otherwise

                (t-ti)Ni,k-1(t)          (ti+k-t)Ni+1,k-1(t)
Ni,k(t) = ---------------------- + ----------------------
              ti+k-1-ti                ti+k-ti+1
```

The curve is evaluated at the points:

```
t =  (1 - i)tk + itj - k + 1
     ------------------------
                q
```

for i=0,1,2,...,q.

PARAMETERS

k — The order of the curve (0 < k).

n — The number of vertices (used to anticipate storage requirements).

x1,y1,z1...xn,yn,zn — The vertices of the defining polygon of the curve. The Z component is optional.

t1,t2,...,tj — User-specified knot sequence. Because closed B-splines are evaluated as open B-splines with duplicate vertices, the number of knots required is:

```
n+k      for open B-splines
n+k+1    for closed nonperiodic B-splines
n+2k-1   for closed periodic B-splines
```

The knots must also be nondecreasing.

q — The number of vectors to be created (0 < q < 32767).

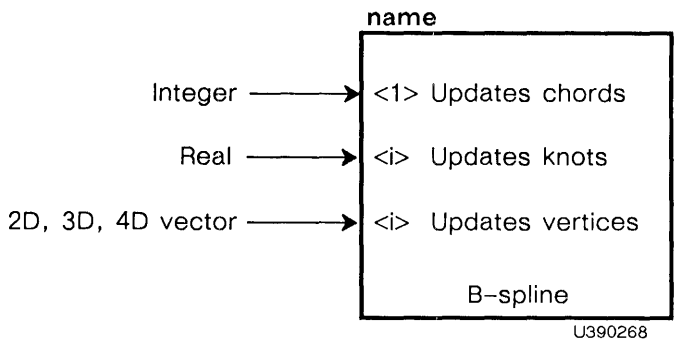
NOTES

1. OPEN or CLOSED is an option which describes the B-spline defining polygon. The default is OPEN. (Note that CLOSED merely describes the polygon, eliminating repetition of the last vertex.)
2. If no knot sequence is given, NONPERIodic or PERIodic is an option which specifies that the nonperiodic or periodic knot sequence be used as the knot sequence. NONPERIodic is the default for open B-splines; PERIodic is the default for closed B-splines.
3. At least k vertices must be given, or the order k will be reduced accordingly.

DISPLAY STRUCTURE NODE CREATED

B-spline vector-list data node.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

1. The Z value of a vector defaults to 0 when a 2D vector is sent to a 3D B-spline.
2. W and Z values will be ignored when a 3D or 4D vector is sent to a 2D B-spline.

CANCEL XFORM

TYPE

MODELING — Transformed Data Attributes

FORMAT

```
name := CANCEL XFORM [APPLied to name1];
```

DESCRIPTION

This command stops transformed data processing of subsequent nodes in a display structure.

PARAMETER

name1 — The node below which to stop transformed data processing

DISPLAY STRUCTURE NODE CREATED

CANCEL XFORM operation node

CHARACTER FONT

TYPE

MODELING — Character Font

FORMAT

```
name := character FONT font_name [APPLied to name1];
```

DESCRIPTION

Establishes a user-defined alternate character font as the working font. This font must have been previously defined with the BEGIN_Font ... END_Font command. If the font is not defined, the current font is still used.

PARAMETERS

font_name — Name of the desired font.

name1 — Structure to use the character font.

DISPLAY STRUCTURE NODE CREATED

Character-font pointer node.

EXAMPLE

```
New_Font := BEGIN_Font
              {character definitions}
            END_Font
A := BEGIN_Structure
        CHARacters 'HERE';  {this uses standard font}
        character FONT New_Font;
        CHARacters 0,-2 'HERE';  {this uses the font New_Font}
     END_Structure;
DISPlay A;
```

CHARACTER ROTATE

TYPE

MODELING — Character Transformations

FORMAT

```
name := CHARacter ROTate angle [APPLied to name1];
```

DESCRIPTION

Rotates characters. Creates a 2x2 rotation matrix to be applied to the specified characters (in name1).

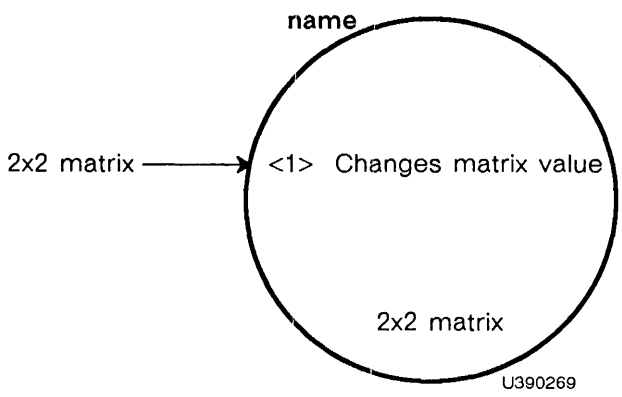
PARAMETER

angle — Z-rotation angle in degrees (unless other units are specified). When you are looking along the positive direction of the Z axis, positive angle values produce counterclockwise rotations.

DISPLAY STRUCTURE NODE CREATED

2x2-matrix operation node.

INPUTS FOR UPDATING NODE



NOTE ON INPUTS

Any 2x2 matrix is legal.

ASSOCIATED FUNCTIONS

F:MATRIX2, F:CROTATE, F:CSCALE

EXAMPLE

```
A:= CHARacter ROTate 90 THEN B;
B:= CHARacters 'Vertical';

{If A were DISPlayed, the text 'Vertical' would start at the origin and
read up the Y axis.}
```

CHARACTERS

TYPE

MODELING — Primitives

FORMAT

```
name := CHARacters [x,y[,z]][STEP dx,dy] 'string';
```

DESCRIPTION

Displays character strings and (optionally) specifies their location and placement.

PARAMETERS

x,y,z — Location in the data space of the beginning of the character string (i.e., the lower left corner of a box enclosing the first character).

dx,dy — Spacing between the characters, in character-size units. The width of the character is one dx unit; the height is one dy unit.

string — Text string to be displayed (up to 240 characters).

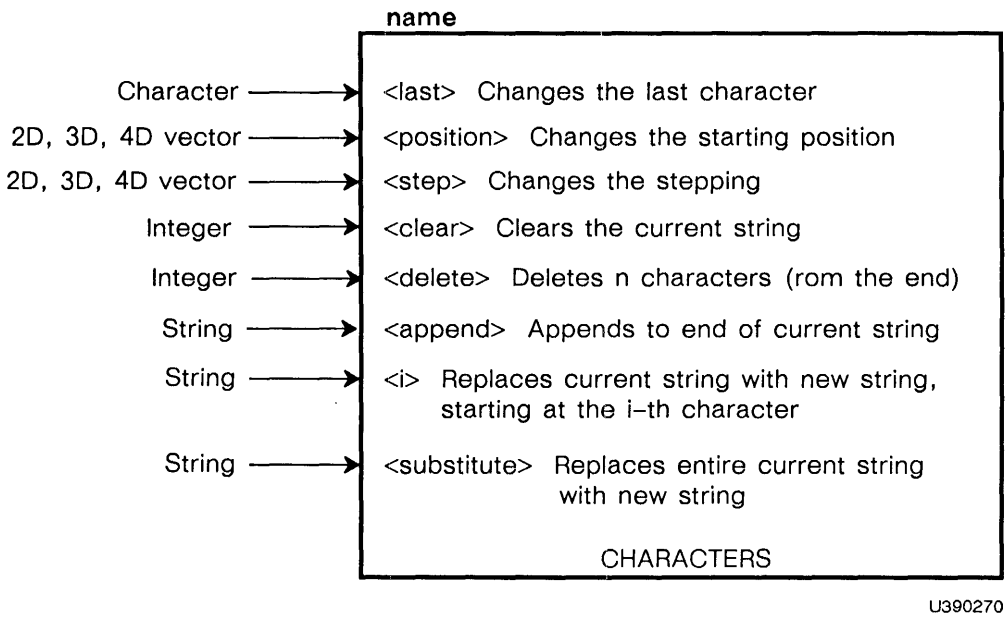
DEFAULT

If string is the only parameter specified, the character string will start at 0,0,0 and dx,dy will be 1,0 (i.e., regular horizontal spacing).

DISPLAY STRUCTURE NODE CREATED

CHARACTERS data node.

INPUTS FOR UPDATING NODE



EXAMPLE

```
CHARacters 'HERE';
CHARacters 3,-3 STEP .5,1 'HERE';
CHARacters STEP -1,0 'HERE';
```

TYPE

MODELING — Character Transformations

FORMAT

```
name := CHARacter SCAle s [APPLied to name1];
name := CHARacter SCAle sx,sy [APPLied to name1];
```

DESCRIPTION

Creates a uniform (s) or nonuniform (sx,sy) 2x2 scaling matrix to scale the specified characters.

PARAMETERS

s — Scaling factor for both axes.

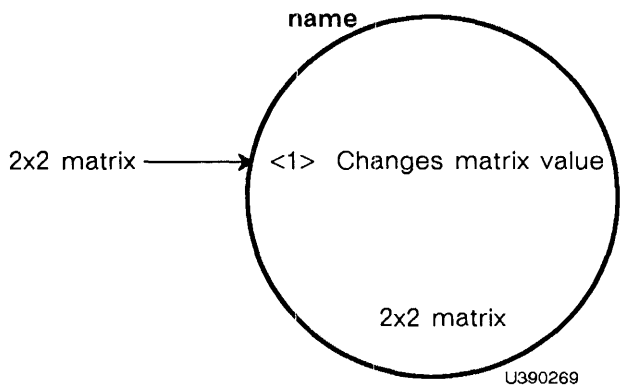
sx,sy — Separate axial scaling factors.

name1 — Structure whose characters are to be scaled (vector lists in the structure are not affected).

DISPLAY STRUCTURE NODE CREATED

2x2-matrix operation node.

INPUTS FOR UPDATING NODE



NOTE ON INPUTS

Any 2x2 matrix is legal.

ASSOCIATED FUNCTIONS

F:MATRIX2, F:CROTATE, F:CSCALE

EXAMPLE

```
A:= CHARacter SCAle .5 THEN B;
B:= CHARacters 'Half scale';
```

COMMAND STATUS

TYPE

GENERAL — Command and Control Status

FORMAT

```
COMmand STATus;
```

DESCRIPTION

Used with BEGIN...END and BEGIN_STRUCTURE...END_STRUCTURE commands to report the current level to which these structures are nested.

NOTES

1. If a syntactically correct command produces a parser syntax error, there may be unENDed BEGINs or BEGIN_Structures causing the PS 390 to expect one or more ENDs or END_Structures. By sending COMmand STATus, you can see if this is the case.
2. The !RESET command can be used to get out of unended BEGINs or BEGIN_Structures when a problem occurs (refer to !RESET).

CONFIGURE

TYPE

GENERAL — Command and Control Status

FORMAT

```
CONFIGURE password;
```

DESCRIPTION

This command allows you to enter the Configure mode (privileged mode). The password can be defined in the SITE.DAT file using the SETUP PASSWORD command. If no password has been set up using SETUP PASSWORD, any string may be used as a password.

PARAMETER

password — the established string

TYPE

FUNCTION — Immediate Action

FORMAT

```
CONNect name1<i>:<j>name2;
```

DESCRIPTION

Connects function instance name1's output <i> to input <j> of function instance or display structure node name2.

PARAMETERS

name1 — Function instance to be connected from.

<i> — Output number of function instance name1 to be connected. Refer to Sections *RM2 Intrinsic Functions* and *RM3 Initial Function Instances* for specific functions and acceptable values.

name2 — Function instance or display structure node to be connected to.

<j> — Input number or input name (in the case of some display structure nodes) of name2 to be connected. Refer to Sections *RM2* and *RM3* for specific functions and acceptable values.

COPY

TYPE

MODELING — Primitives

FORMAT

```
name := COPY name1 [START=] i [,] [COUNT=] n;
```

DESCRIPTION

Creates a VECtor_list node containing a group of consecutive vectors copied from another vector list (name1) or a LABELS node containing a group of consecutive labels from an existing block (name1).

PARAMETERS

name — Name of new VECtor_list or LABELS node.

name1 — Name of the node being copied from.

i — First vector or index of first label in name1 to be copied.

n — Last vector or count of labels in name1 to be copied.

NOTE

The keywords START= and COUNT= are optional, but if one is used, both must be used.

DISPLAY STRUCTURE NODE CREATED

VECtor_list or LABELS data node.

INPUTS FOR UPDATING NODE

(Refer to VECtor_list or LABELS command).

EXAMPLE

```
A := VECtor_list n=5 .5,.5 -.5,.5 -.5,-.5 .5,-.5 .5,.5;
B := COPY A 1 3;

    {This would be the same as saying:
     B := VECtor_list n=3 .5,.5 -.5,.5 -.5,-.5;}


C := COPY A START=2 , COUNT=2;

    {This would be the same as saying:
     C := VECtor_list n=2 -.5,.5 -.5,-.5;}
```

DECREMENT LEVEL_OF_DETAIL

TYPE

STRUCTURE — Attributes

FORMAT

```
name := DECrement LEVel_of_detail [APPLied to name1];
```

DESCRIPTION

"Decrements" (decreases) the current level of detail by 1 when name is being traversed.

PARAMETER

name1 — Structure to be affected by the decreased level of detail.

NOTE

There is really only one global level of detail; this command only changes the value of the level of detail while the named node and nodes below it in a display structure are being traversed.

DISPLAY STRUCTURE NODE CREATED

DECrement LEVel_of_detail operation node.

EXAMPLE

```
A:= SET LEVel_of_detail TO 5 THEN B;
B:= BEGIN_Structure
    IF LEVel_of_detail = 4 THEN C;
    IF LEVel_of_detail = 5 THEN D;
    DECrement LEVel_of_detail;
    IF LEVel_of_detail = 4 THEN E;
    IF LEVel_of_detail = 5 THEN F;
    END_Structure;

{If A were DISPlayed, structures D and E would also be displayed.}
```

TYPE

GENERAL — Data Structuring and Display

FORMAT

```
DELete name[,name1 ... namen];
DELete any_string*;
```

DESCRIPTION

Sets name to nil, then FORGETs name. The wild card delete will set to nil any name beginning with the string that is entered.

PARAMETERS

name — Any previously defined name.

any_string — A character string which is part of any name.

NOTES

1. After a DELete name command is issued, all function instances and structures referring to name will no longer include the data formerly associated with name.
2. After a DELete name command is issued, further definitions of or references to name will not change structures which referred to name before the DELete.
3. Compare with FORGET, which eliminates name while preserving objects which it formerly referred to.
4. If the wild card delete is used on an object being displayed, the object must be removed from display before entering the wild card delete command. Failure to do this will result in a small amount of memory being used for each object still displayed.
5. If a name is created from the host, it must be deleted via the host line. Similarly, if a name is created locally using the keyboard, the DELete command must be entered locally.

DISCONNECT

TYPE

FUNCTION — Immediate Action

FORMAT

```
DISCONNect name1[<i>]:option;
DISCONNect name1<i>:<j>name2;
```

DESCRIPTION

Disconnects one or all of the outputs of function instance name1 from one or all inputs that it has previously been connected to.

PARAMETERS

name1 — Function instance to disconnect output(s) from.

<i> — The output number of name1 to disconnect. If this is not specified, all of name1's outputs are implied and the option parameter must be ALL (this would disconnect all of name1's outputs from everything they had previously been connected to).

ALL — Disconnect the specified output of name1 (or all outputs of name1) from all function instances or display structure nodes that it was previously connected to.

<j> — Input number or input name of name2 to be disconnected from name1.

name2 — Function instance or named node previously connected to name1.

DISPLAY

TYPE

GENERAL — Data Structuring and Display

FORMAT

```
DISPlay name;
```

DESCRIPTION

Displays a structure. Adds name to the display processor's display list.

PARAMETER

name — Any structure name.

ERASE PATTERN FROM

TYPE

MODELING — Primitives

FORMAT

```
ERASE PATTERN FROM name;
```

DESCRIPTION

An immediate-action command which erases a pattern from a vector list (name).

PARAMETER

name — The vector list containing the pattern you want to erase.

EYE BACK

TYPE

VIEWING — Windowing Transformations

FORMAT

```
name := EYE BACK z [option1][option2] from SCREEN area w WIDE
        [FRONT boundary = zmin BACK boundary = zmax]
        [APPLied to name1];
```

DESCRIPTION

Specifies a viewing pyramid with the eye at the apex and the frustum of the pyramid (bounded by zmin and zmax) enclosing a portion of the data space to be displayed in perspective projection. Unlike the Field_Of_View command, the EYE BACK command can create a skew (nonright) viewing pyramid (compare Field_Of_View and WINDOW).

PARAMETERS

z — The perpendicular distance of the eye from the plane of the viewport.

option1 — RIGHT X or LEFT X, where X is the distance of the eye right or left of the viewport center, respectively, in relative room coordinates.

option2 — UP Y or DOWN Y, where Y is the distance of the eye up or down from the viewport center, respectively, in relative room coordinates.

w — Width of the viewport in relative room coordinates.

zmin,zmax — Front and back boundaries of the frustum of the viewing pyramid. (Refer to note 3 of the LOOK command for properly specifying zmin and zmax.)

name1 — Structure to which the EYE BACK viewing area is applied.

DEFAULT

None. If no EYE BACK is specified, the default WINDOW is assumed (parallel projection X = -1:1 Y = -1:1 FRONT = 10E-15 BACK = 10E+15). Refer to the WINDOW command.

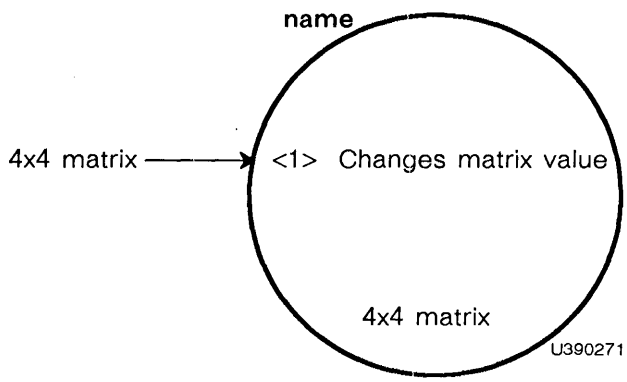
NOTES

1. Notice that EYE BACK always creates square side boundaries because the viewport width (w) is also taken to be the height; the aspect ratio is always 1.
2. If X and Y are not specified (i.e. 0), then a right rectangle viewing pyramid is created (compare Field_Of_View).

DISPLAY STRUCTURE NODE CREATED

4x4-matrix operation node.

INPUT FOR UPDATING NODE



ASSOCIATED FUNCTIONS

F:FOV, F:WINDOW, F:MATRIX4

EXAMPLE

```
A:= BEGIN_Structure
    EYE BACK 24 LEFT 1.5 from SCREEN area 10 WIDE
         FRONT boundary = 12
         BACK boundary = 14;
    LOOK AT 0,0,0 FROM 5,6.63,-10;
    INSTance of sphere;
    END_Structure;

{If sphere is defined with a radius of 1 about the origin, A would be a
view of the sphere from 5, 6.63, -10 fully depth cued. Note that the
FROM to AT distance in the LOOK AT command is 13.}
```

FIELD_OF_VIEW

TYPE

VIEWING — Windowing Transformations

FORMAT

```
name := Field_Of_View angle
            [FRONT boundary = zmin BACK boundary = zmax]
            [APPLied to name1];
```

DESCRIPTION

Specifies a right-rectangular viewing pyramid with the eye at the apex and the frustum of the pyramid (bounded by zmin and zmax) enclosing a portion of the data space to be displayed in perspective projection (compare EYE and WINDOW).

PARAMETERS

angle — Angle of view from the eye (i.e., the FROM point established in the LOOK command) in X and Y. (Refer to note 1 below.)

zmin,zmax — Front and back boundaries of the frustum of the viewing pyramid. (Refer to note 3 of the LOOK command for properly specifying zmin and zmax.)

name1 — Structure to which the FOV is applied.

DEFAULT

None. If no Field_Of_View is specified, the default WINDOW is assumed instead (parallel projection X = -1:1 Y = -1:1 FRONT = 10E-15 BACK = 10E+15). Refer to the EYE command.

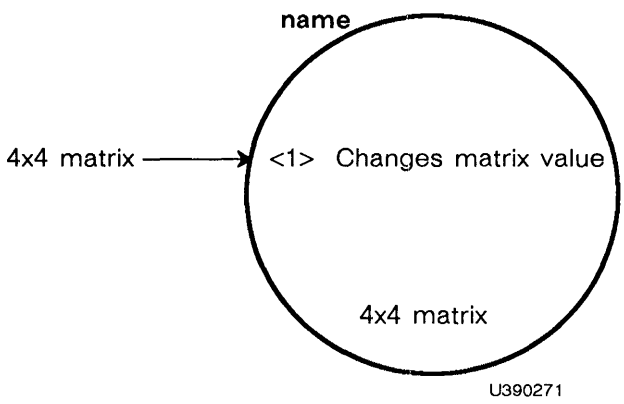
NOTES

1. Notice that FOV always creates square side boundaries because angle defines both the X and the Y angles; the aspect ratio is always 1.
2. Refer also to notes for the WINDOW command.

DISPLAY STRUCTURE NODE CREATED

4x4-matrix operation node.

INPUT FOR UPDATING NODE



ASSOCIATED FUNCTIONS

F:FOV, F:WINDOW, F:MATRIX4

EXAMPLE

```
BEGIN_Structure
    Field_Of_View 30
        FRONT boundary 12
        BACK boundary 14;
        LOOK AT 0,0,0 FROM 5,6.63,-10;
        INSTance of Sphere;
END_Structure;

{If Sphere is defined with a radius of 1 about the origin, A would be a
view of the Sphere from 5, 6.63, -10 fully depth cued. Note that the
FROM to AT distance in the LOOK command is 13.}
```

FINISH CONFIGURATION

TYPE

GENERAL — Command and Control Status

FORMAT

```
FINISH CONFIGURATION;
```

DESCRIPTION

This command takes the PS 390 out of Configure mode and must be used at the end of any session that has modified any part of the CONFIG.DAT file or accessed any system-level functions.

FOLLOW WITH

TYPE

STRUCTURE — Modifying

FORMAT

```
FOLLOW name WITH option;
```

DESCRIPTION

Follows a named operation node (**name**) with another operation node.

PARAMETERS

name — A named transformation, attribute, or conditional reference node to be followed with one of the options.

option —

1. A node created by a transformation command (SCALE by, ROTate, etc).
2. A node created by an attribute-setting command (SET LEVel_of_detail, etc.).
3. A node created by a conditional-referencing command (IF LEVel_of_detail, etc).

NOTE

The structure **name** does not change association, unlike a named structure in a PREFIX WITH command.

DISPLAY STRUCTURE NODE CREATED

An operation node corresponding to the option phrase of the command. This node points to whatever node name pointed to previously. The node is also pointed to by name.

EXAMPLE

```
Shape :=  BEGIN_Structure
          Tran := TRANslate by 20,20;
          Rotate := ROTate in X 90;
          Triangle := VECtor_list n=4 0,0 0,3 3,0 0,0;
          END_Structure;
FOLLOW Shape.Rot WITH SCALE by 2;

{This will alter the structure Shape so that Shape.Triangle is first
scaled, then rotated, then translated.}
```

FORGET (Structures)

TYPE

GENERAL — Data Structuring and Display

FORMAT

```
FORget name;
```

DESCRIPTION

Removes **name** from the display (if **name** is being displayed), and removes **name** from the Name dictionary.

PARAMETER

name — Any previously defined structure name.

NOTES

1. After a FORget name command is issued for a structure, all function instances and structures referring to name will continue to refer to the data formerly associated with name, even though name is no longer linked with the data.
2. After a FORget name command is issued for a structure, further definitions of, or references to, name will not change structures which referred to name before the FORget command.
3. Compare with DELete, which affects not only name but the content of name also.

TYPE

GENERAL — Data Structuring and Display

FORMAT

```
FORget (unit_name);
```

DESCRIPTION

Removes a unit definition from memory.

PARAMETER

unit_name — Any previously assigned unit name.

NOTE

Note that FORget requires unit names to be enclosed in parentheses (unlike structure names).

(Function Instancing)

TYPE

STRUCTURE — Explicit Referencing

FORMAT

```
NAME := F:function_name;
```

DESCRIPTION

Creates an instance of a PS 390 intrinsic function.

PARAMETERS

name — Any combination of alphanumeric characters up to 240. Must begin with an alpha character and can include $ or _.

function_name — Any PS 390 intrinsic function name.

EXAMPLE

```
Add1 := F:ADD;
Add2 := F:ADD;

{This creates two different instances of the same Intrinsic Function
F:ADD.}
```

GIVE_UP_CPU

TYPE

GENERAL — Command and Control Status

FORMAT

```
GIVE_UP_CPU;
```

DESCRIPTION

This command causes the command interpreter to terminate execution temporarily and allow other functions to be activated.

NOTE

To ensure that other functions are activated, the GIVE_UP_CPU command should be sent four times after sending a value to F:ALLOW_VECNORM.

IF CONDITIONAL_BIT

TYPE

STRUCTURE — Conditional Referencing

FORMAT

```
name := IF conditional_BIT n is state [THEN name1];
```

DESCRIPTION

Refers to a structure if an attribute bit has a specified setting (ON or OFF). (Refer to SET conditional_BIT command.)

PARAMETERS

n — Integer from 0 to 14 indicating which bit to test.

state — The setting to be tested (ON or OFF).

name1 — Structure to be conditionally referenced.

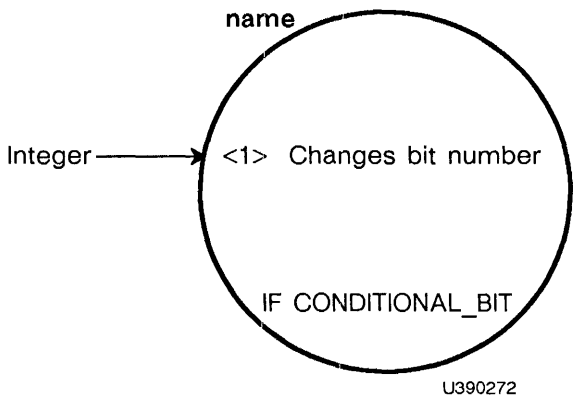
DEFAULT

If bit n was not manipulated higher in the display structure, it will default to OFF.

DISPLAY STRUCTURE NODE CREATED

IF conditional_BIT operation node (conditional connection between two structures).

INPUT FOR UPDATING NODE



NOTE ON INPUT

Input <1> accepts an integer (between 0 and 14) to change the bit number to the integer value.

EXAMPLE

```
A:= SET conditional_BIT 3 ON THEN B;
B:= IF conditional_BIT 3 is ON THEN C;
C:= VECtor_list ... ;

{Initially when A is DISPlayed, C would also be displayed, indirectly.
If a function network were connected to A to change conditional bit 3
to OFF, then the test in B would fail and C would not be displayed.}
```

IF LEVEL_OF_DETAIL

TYPE

STRUCTURE — Conditional Referencing

FORMAT

```
name := IF LEVel_of_detail relationship n [THEN name1];
```

DESCRIPTION

Refers to a structure if the level of detail attribute has a specified relationship to a given number. Tests the relation between the current level of detail and the number n (see SET LEVel_of_detail command).

PARAMETERS

relationship — The relationship to be tested (<, <=, =, <>, >=, >).

n — Integer from 0 to 32767 indicating the number to compare the current level of detail to.

name1 — Structure to be conditionally referenced.

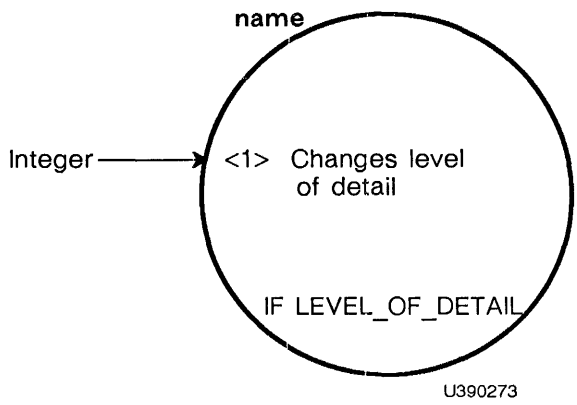
DEFAULT

If the level of detail is not manipulated higher in the structure by a SET LEVel_of_detail node, it will default to 0.

DISPLAY STRUCTURE NODE CREATED

IF LEVel_of_detail operation node (conditional connection between two structures).

INPUT FOR UPDATING NODE



NOTE ON INPUT

Input <1> accepts an integer (from 0 to 32767) to change the level of detail to the integer value.

EXAMPLE

```
A:= SET LEVel_of_detail to 3 THEN B;
B:= IF LEVel_of_detail = 3 THEN C;
C:= VECtor_list ... ;

{Initially when A is DISPlayed, C would also be displayed, indirectly.
If a function network were connected to A to change the level of detail
to something other than 3, then the test in B would fail and C would
not be displayed.}
```

IF PHASE

TYPE

STRUCTURE — Conditional Referencing

FORMAT

```
name := IF PHASE is state THEN [name1];
```

DESCRIPTION

Refers to a structure if the PHASE attribute is in a specified state (ON or OFF). (Refer to SET RATE and SET RATE EXTernal commands.)

PARAMETERS

state — Phase setting to be tested (ON or OFF).

name1 — Structure to be conditionally referenced.

DEFAULT

If there is no SET RATE node or SET RATE EXTernal node higher in the display structure, the PHASE attribute will always be OFF.

DISPLAY STRUCTURE NODE CREATED

IF PHASE operation node (conditional connection between two structures).

EXAMPLE

```
A:= SET RATE 10 15 THEN B;
B:= IF PHASE is ON THEN C;
C:= VECtor_list ... ;

{If A is DISPlayed, C will also be displayed for 10 refresh frames and
not DISPlayed for 15 refresh frames repetitively.}
```

ILLUMINATION

TYPE

RENDERING — Data Structuring

FORMAT

```
name := ILLUMINATION x,y,z [COLOR h [,s [,i]]] [AMBIENT a];
```

DESCRIPTION

Specifies light sources for shaded images created with the PS 390 rendering firmware option. An unlimited number of light sources may be specified. For a detailed explanation of defining and interacting with shaded images, consult Section *GT13 Polygonal Rendering*.

PARAMETERS

x,y,z — A vector from the origin pointing towards the light source.

h — A real number specifying the hue in degrees around the color wheel. Pure blue is 0 and 360, pure red is 120, and pure green is 240.

s — A real number specifying saturation. No saturation (gray) is 0 and full saturation (full toned colors) is 1.

i — A real number specifying intensity. No intensity (black) is 0, full intensity (white) is 1.

a — A real number which controls the contribution of a light source to the ambient light. Increasing **a** for a light source increases its contribution to the ambient light.

DEFAULT

If no ILLUMINATION command is used, a default white light at (0,0,-1) with an ambient proportion of 1.0 is assumed. If intensity and saturation are not specified, they default to 1. If only hue and saturation are specified, intensity defaults to 1. The default for ambient proportion is 1.

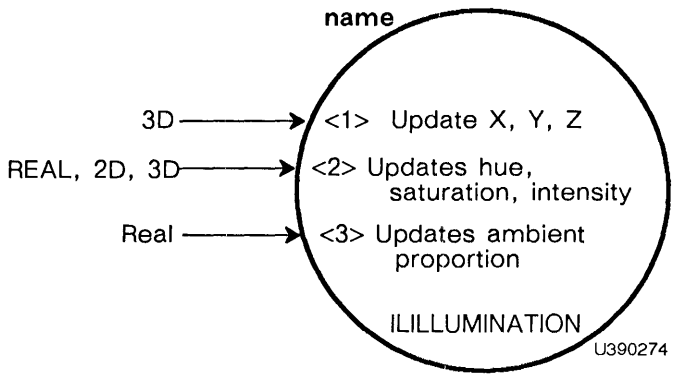
NOTES

1. Illumination nodes may be placed anywhere in a display structure, allowing lights to be stationary or to rotate with the object, or both.
2. An unlimited number of light sources are valid for smooth-shaded renderings, but only the last illumination node encountered is used in creating flat-shaded renderings.
3. Light sources are not used in wash-shaded (area-filled) images.

DISPLAY STRUCTURE NODE CREATED

ILLUMINATION operation node.

INPUTS FOR UPDATING NODE



NOTE ON INPUTS

A real number sent to input <2> changes only the hue. In this case, saturation and intensity default to 1. You cannot change just one value and retain the remaining values. Unless a 3D vector is sent, the default values are assumed for the variables not specified.

EXAMPLE

```
Light := ILLUMINATION 1,1,-1 COLOR 180;

{This creates a node which defines a yellow light over the right
shoulder. Since saturation and intensity are not specified, the
defaults s = 1 and i = 1 are assumed. The ambient proportion defaults
to 1.}
```

TYPE

STRUCTURE — Modifying

FORMAT

```
INCLude name1 IN name2;
```

DESCRIPTION

Used to include (instance) another named entity (name1) under a named instance node in a display structure (name2).

PARAMETERS

name1 — Structure to be included under instance node name2.

name2 — Name of the instance node to include name1.

DISPLAY STRUCTURE NODE CREATED

None. This is an immediate-action command which modifies an existing instance node in a display structure.

EXAMPLE

```
Map:= INSTance of Canada, South_America, United_States;
INCLude Mexico IN Map;

{This would result in the instance node called Map also pointing at
Mexico.}
```

INCREMENT LEVEL_OF_DETAIL

TYPE

STRUCTURE — Attributes

FORMAT

```
NAME := INCRement LEVel_of_detail[APPLied to name1];
```

DESCRIPTION

"Increments" (increases) the current level of detail by 1 when name is being traversed.

PARAMETER

name1 — Node to be affected by the increased level of detail.

NOTE

There is really only one global level of detail; this command only changes the value of the level of detail while the named node and nodes below it in the display structure are being traversed.

DISPLAY STRUCTURE NODE CREATED

INCRement LEVel_of_detail operation node.

EXAMPLE

```
A:= INCRement LEVel_of_detail THEN B;
B:= INSTance of C, D;
C:= IF LEVel_of_detail = 1 THEN E;
D:= IF LEVel_of_detail = 2 THEN F;

{If A were DISPlayed, E would also be displayed but not F. Since the
default level of detail is 0, A will change the level of detail to 1,
so the test in C will pass to E, while the test in D will fail and F
will not be traversed.}
```

INITIALIZE

TYPE

GENERAL — Initialization

FORMAT

```
INITialize [option];
```

DESCRIPTION

INITialize (without specifying an option) restores the PS 390 to its initial state in which:

- No user-defined names exist.
- No user-defined units exist.
- No user-created display structures exist.
- No user-defined function connections exist.
- No structures are being displayed.

You may also initialize any of the above areas selectively (without initializing others) by following INITialize with the appropriate keyword for the area to be initialized.

The INITialize command also automatically executes the OPTIMIZE MEMORY command to collect any contiguous free blocks of memory into single blocks.

PARAMETERS

option — Any of the following:

CONNections — Breaks all user-defined function connections.

DISPlay — Removes all structures from the display list.

NAMES — Clears the name dictionary of all structures and function instance names.

UNITS — Clears all user-defined units.

NOTES

1. An INITialize command is specific to a command interpreter. It only affects the structures which were established by the same command interpreter as the initialization command itself. For example, structures created through the host line can be removed with an INITialize from the host, but not by an INITialize from the PS 390 keyboard.
2. The INITialize command blanks every object being displayed whether the object was created from the host or locally.

TYPE

STRUCTURE — Explicit Referencing

FORMAT

```
name := INSTance of name1[,name2...,namen];
```

DESCRIPTION

Groups one or more structures under a single named instance node.

PARAMETERS

name1...namen — Structures to be grouped.

DISPLAY STRUCTURE NODE CREATED

An instance node with pointers to each of the structures referenced (name1...namen).

INPUTS FOR UPDATING NODE

None; however the INCLude and REMove commands can be used to modify the instance node.

EXAMPLE

```
A:= INSTance of B,C,D;
```

LABELS

TYPE

MODELING — Primitives

FORMAT

```
name := LABELS    x,y [,z] 'string'
                        .
                        .
                  [xi,yi [,zi] 'string'];
```

DESCRIPTION

The LABELS command, like CHARacters, defines character strings for display. However, a single LABELS command can define an indefinitely large number of character strings.

PARAMETERS

x,y,z — Coordinates of the lower left-hand corner of the first character in the string.

string — Text string up to 240 characters in length.

DEFAULT

If z is not specified, it is assumed to be 0.

NOTES

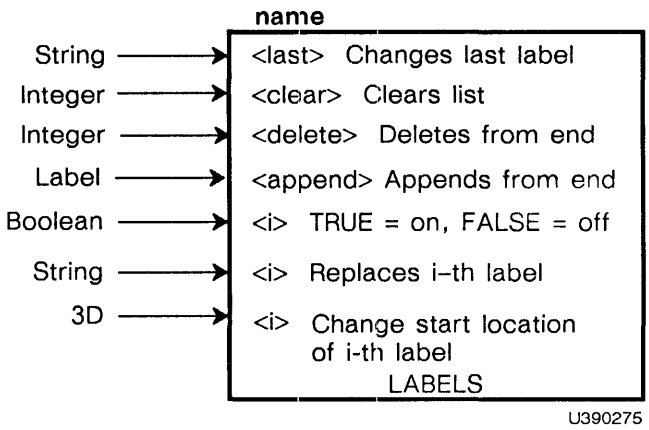
1. A gain in display capacity is realized whenever two or more character strings are combined in a single LABELS command.
2. The smallest LABELS entity that can be picked is an entire string; a pick returns an index into the list of strings of the LABELS command. Individual characters cannot be picked as they can with CHARacters.
3. The commands SET CHARacters SCREEN_oriented/[FIXED] and SET CHARacters WORLD_oriented can be applied to LABELS in the same way they are applied to CHARacters.

4. You may SEND messages to a LABELS node as you can to a CHARacters node.

DISPLAY STRUCTURE NODE CREATED

LABELS data node.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

1. Sending an integer to <delete>of a LABELS node deletes that many strings from the end of the labels block. If the integer is as large as or larger than the number of strings in the block, then all strings are removed except the first. This is retained to keep the step size information, but display of that string is disabled.
2. Sending an integer to <clear> of a LABELS node deletes all labels except the first, which is retained for step size information, but is not displayed.
3. The <append> input accepts only special "label"-type messages that give both the string and the position to be appended. This data type is created by the F:LABEL function.

EXAMPLE

```
A:= LABELS 0,0 'FIRST LINE'
    0,-1.5 'SECOND LINE';
```

LOAD VIEWPORT

TYPE

VIEWING — Viewport Specification

FORMAT

```
name := LOAD VIEWport HORizontal = hmin:hmax
              VERTical = vmin:vmax
              [INTENsity = imin:imax] [APPLied to name1];
```

DESCRIPTION

The LOAD VIEWPORT command loads a viewport and overrides the concatenation of the previous viewport. As with the standard PS 390 VIEWPORT command, it specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines. It affects all objects below the node created by the command in the display structure.

PARAMETERS

hmin,hmax,vmin,vmax — The x and y boundaries of the new viewport. Values must be within the -1 to 1 range.

imin,imax — Specifies the minimum and maximum intensities for the viewport. imin is the intensity of lines at the back clipping plane; imax at the front clipping plane. Values must be within the 0 to 1 range.

name1 — The name of the structure to which the viewport is applied.

DEFAULT

The initial viewport is the full PS 390 screen with full intensity range (0 to 1) using the standard PS 390 VIEWPORT command.

```
VIEWport HORizontal = -1:1 VERTical = -1:1 INTENsity = 0:1;
```

NOTES

1. A new VIEWport is not defined relative to the current viewport, but to the full PS 390 screen.

2. If the viewport aspect ratio (vertical/horizontal) is different from the window aspect ratio (y/x) or field-of-view aspect ratio (always 1) being displayed in that viewport, the data displayed there will appear distorted.

DISPLAY STRUCTURE NODE CREATED

This command creates a load viewport operation node that has the same inputs as the standard viewport operation node. The matrix contained in this node is not concatenated with the previous viewport matrix.

NOTES ON INPUTS

1. For 2x2-matrix input, row 1 contains the **hmin,hmax** values and row 2 the **vmin,vmax** values.
2. For 3x3-matrix input, column 3 is ignored (there is no 3x2-matrix data type), rows 1 and 2 are as for the 2x2 matrix above, and row 3 contains the **imin,imax** values.

TYPE

VIEWING — Windowing Transformations

FORMAT

```
name := LOOK AT ax,ay,az FROM fx,fy,fz
             [UP ux,uy,uz] [APPLied to name1];
name := LOOK FROM fx,fy,fz AT ax,ay,az
             [UP ux,uy,uz] [APPLied to name1];
```

DESCRIPTION

This command, in conjunction with a windowing command (WINDOW, Field_Of_View, or EYE), fully specifies the portion of the data space that will be viewed, as well as the viewer's own orientation in the world coordinate system.

The LOOK AT...FROM clauses specify the viewer's position with respect to the object(s), while the optional UP clause specifies the screen "up" direction (analogous to adjusting the way the viewer's head is tilted).

LOOK creates a 4x3 transformation matrix which:

1. Translates the data base so that the FROM point is at the origin (0,0,0).
2. Rotates the data base so that the AT point is along the positive Z axis at (0,0,D), where D = || F–A ||.
3. Rotates the data base so that the UP vector is in the YZ plane.

PARAMETERS

ax,ay,az — Point being looked at, in world coordinates.

fx,fy,fz — Location of viewer's eye, in world coordinates.

ux,uy,uz — Vector indicating screen "up" direction.

name1 — Any structure.

DEFAULT

```
LOOK AT 0,0,1 FROM 0,0,0 UP 0,1,0;
```

NOTES

1. To be implemented properly in a display structure, the LOOK node must follow one of the windowing nodes and may not precede any windowing node. (Refer to WINDOW Notes.)
2. The UP vector indicates a direction only; its magnitude does not matter. For example, the two clauses UP 0,1,0 and UP 0,10,0 have exactly the same effect.
3. In determining FRONT and BACK boundary parameters for an associated windowing command (WINDOW, Field_Of_View, or EYE), remember that the LOOK command positions the AT point along the positive Z axis at 0,0,D where D equals the distance of the FROM point to the AT point. So, for example, if the FROM to AT distance is 13, if full depth cueing is desired, and the radius of the object is 1, then

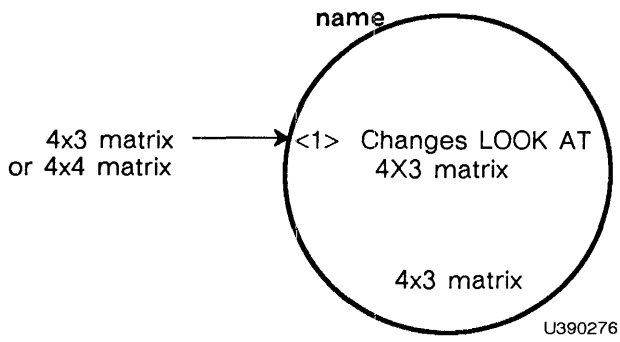
   ```
   FRONT boundary = 12
   BACK boundary = 14
   ```

   is used.

DISPLAY STRUCTURE NODE CREATED

4x3-matrix operation node.

INPUT FOR UPDATING NODE



NOTE ON INPUT

If a 4x4 matrix is input, the 4th column is ignored.

ASSOCIATED FUNCTIONS

F:LOOKAT, F:LOOKFROM

EXAMPLE

```
A:= BEGIN_Structure
    WINDOW X = -1:1 Y = -1:1
        FRONT boundary = 12
        BACK boundary = 14;
        LOOK AT 0,0,0 FROM 5,6.63,-10 THEN Sphere;
    END_Structure;

{If Sphere is defined with a radius of 1 about the origin, A would be a
view of the sphere from 5, 6.63, -10, fully depth cued. Note that the
FROM to AT distance in the LOOK command is 13.}
```

TYPE

MODELING — Character Transformations

FORMAT

```
name := Matrix_2x2 m11,m12
                   m21,m22 [APPLied to name1];
```

DESCRIPTION

Creates a 2x2 transformation matrix which applies to characters in the structure that follows (name1).

PARAMETERS

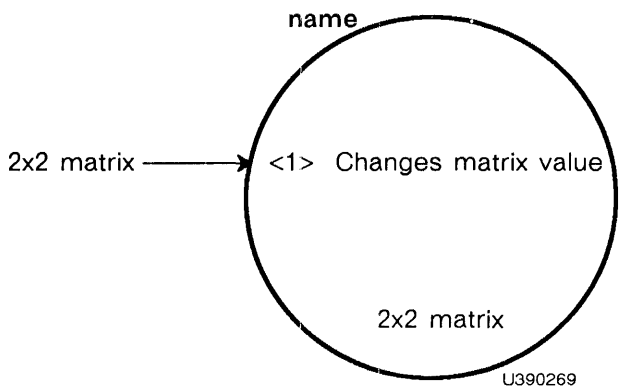
m11 - m22 — Elements of the 2x2 matrix.

name1 — Structure whose characters are to be transformed (any vector lists in the display structure are left unchanged).

DISPLAY STRUCTURE NODE CREATED

2x2-matrix operation node.

INPUT FOR UPDATING NODE



NOTE ON INPUT

Any 2x2 matrix is legal.

ASSOCIATED FUNCTIONS

F:Matrix2, F:CSCALE, F:CROTATE

EXAMPLE

```
A := Matrix_2x2  1,0
                .5,1 THEN B;

{This creates a skewing matrix which is useful for italicizing text.}
```

TYPE

MODELING — Transformations

FORMAT

```
name := Matrix_3x3 m11,m12,m13
                   m21,m22,m23
                   m31,m32,m33 [APPLied to name1];
```

DESCRIPTION

Creates a 3x3 transformation matrix which applies to the specified data (vector lists and/or characters).

PARAMETERS

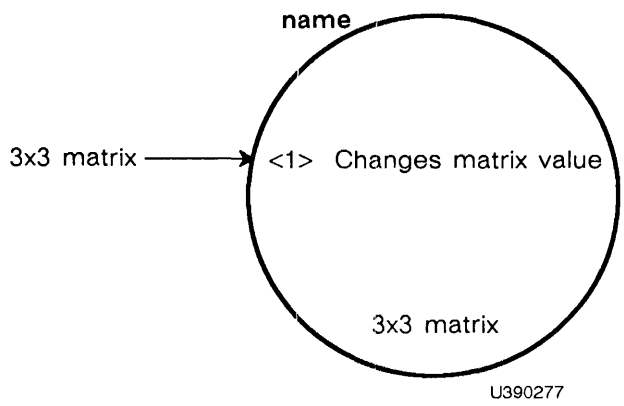
m11 - m33 — Elements of the 3x3 matrix to be created.

name1 — Structure to be transformed by the matrix.

DISPLAY STRUCTURE NODE CREATED

3x3-matrix operation node.

INPUT FOR UPDATING NODE



NOTE ON INPUT

Any 3x3 matrix is legal (a rotation matrix, a scale matrix, etc.).

ASSOCIATED FUNCTIONS

F:MATRIX3, F:XROTATE, F:YROTATE, F:ZROTATE, F:DXROTATE, F:DYROTATE, F:DZROTATE, F:SCALE, F:DSCALE

EXAMPLE

```
A := Matrix_3x3 1,0,0
                0,1,0
                0,0,1 APPLied TO B;

{This creates an identity matrix.}
```

TYPE

MODELING — Transformations

FORMAT

```
name := Matrix_4x3 m11,m12,m13
                   m21,m22,m23
                   m31,m32,m33
                   m41,m42,m43 [APPLied to name1];
```

DESCRIPTION

Creates a 4x3 transformation matrix which applies to the specified data (vector lists and/or characters).

PARAMETERS

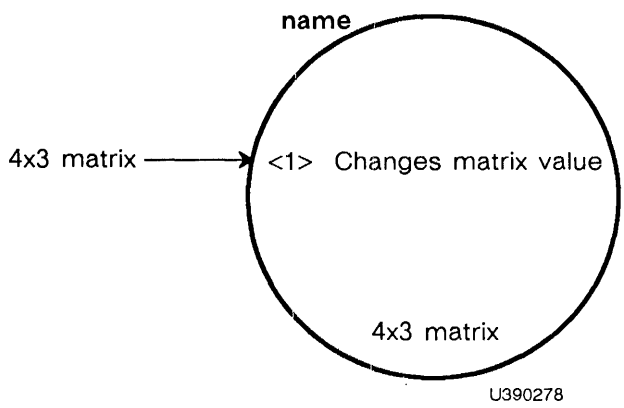
m11 - m43 — Elements of the 4x3 matrix to be created.

name1 — Structure to be transformed by the matrix.

DISPLAY STRUCTURE NODE CREATED

4x3-matrix operation node.

INPUT FOR UPDATING NODE



NOTE ON INPUT

Any 4x3 matrix is legal (a rotation matrix, a scale matrix, etc.).

ASSOCIATED FUNCTIONS

F:MATRIX4, F:XROTATE, F:YROTATE, F:ZROTATE, F:DXROTATE, F:DYROTATE, F:DZROTATE, F:SCALE, F:DSCALE

EXAMPLE

```
A := Matrix_4x3 1,0,0
                0,1,0
                0,0,1
                0,0,0 APPLied TO B;
```

TYPE

MODELING — Transformations

FORMAT

```
name := Matrix_4x4 m11,m12,m13,m14
                   m21,m22,m23,m24
                   m31,m32,m33,m34
                   m41,m42,m43,m44 [APPLied to name1];
```

DESCRIPTION

Creates a 4x4 transformation matrix which applies to the specified data (vector lists and/or characters).

PARAMETERS

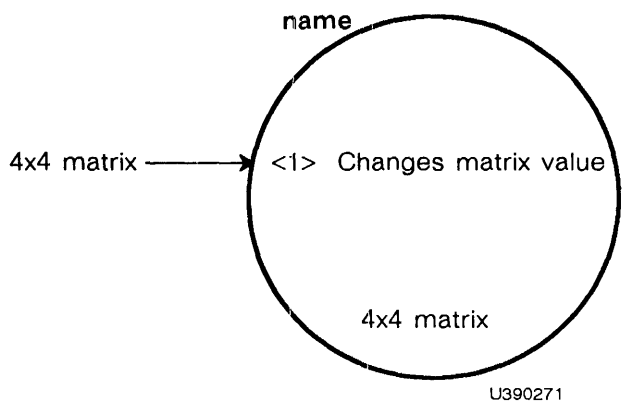
m11 - m44 — Elements of the 4x4 matrix to be created.

name1 — Structure to be transformed by the matrix.

DISPLAY STRUCTURE NODE CREATED

4x4-matrix operation node.

INPUT FOR UPDATING NODE



NOTE ON INPUT

Any 4x4 matrix is legal (a rotation matrix, a scale matrix, etc.).

ASSOCIATED FUNCTIONS

F:MATRIX4, F:XROTATE, F:YROTATE, F:ZROTATE, F:DXROTATE, F:DYROTATE, F:DZROTATE, F:SCALE, F:DSCALE

EXAMPLE

```
A := Matrix_4x4 1,0,0,0
                0,1,0,0
                0,0,1,0
                0,0,0,1 APPLied TO B;

{This creates an identity matrix.}
```

(Naming of Display Structure Nodes)

TYPE

STRUCTURE — Explicit Referencing

FORMAT

```
NAME := display_structure_command;
```

DESCRIPTION

Gives a name (address) to a node in a display structure so that it can be referenced explicitly.

PARAMETERS

name — Any combination of alphanumeric characters up to 240. Must begin with an alpha character and can include $ and _.

display_structure_command — All data structuring commands except the function instancing command (name := F:function_name).

NOTES

1. All nodes in a display structure must be named (addressed) either directly, using this structure naming command, or indirectly, nesting a display structure command within a BEGIN_Structure...END_Structure command.
2. Upper and lowercase letters can be used in names, but all letters are converted to uppercase. Thus turbine_blade, Turbine_Blade, and TURBINE_BLADE are equivalent names.
3. A null structure can be named using the name := NIL; form of the command. If this command were used to redefine name, name would be kept in the name dictionary but the definition previously associated with name would be removed. FORGET name does just the opposite (refer to FORGET). DELETE name removes both the name and its definition (refer to DELETE).

TYPE

STRUCTURE — Explicit Referencing

FORMAT

```
name := NIL;
```

DESCRIPTION

This command names a null data structure. When this command is used to redefine **name, name** is kept in the name dictionary but any data structures previously associated with it are removed. FORGET does just the opposite of NIL.

OPTIMIZE MEMORY

TYPE

GENERAL — Command Control and Status

FORMAT

```
OPTIMIZE MEMORY;
```

DESCRIPTION

An immediate-action command which collects any contiguous free blocks of memory into single blocks.

NOTES

1. If you are transmitting a large vector list from the host and you suspect that memory is being fragmented, enter this command before doing any operations.
2. This command is executed automatically whenever an INITialize command is entered.

OPTIMIZE STRUCTURE;...END OPTIMIZE;

TYPE

GENERAL — Command Control and Status

FORMAT

```
OPTIMIZE STRUCTURE;
      command;
      command;
        .
        .
END OPTIMIZE;
```

DESCRIPTION

Places the PS 390 in, and removes it from, "optimization mode," during which certain elements of a display structure are created in a way that minimizes display processor traversal time.

NOTES

1. Optimization mode is intended for application programs whose development is complete. Since optimization severely restricts the kinds of changes that may be made to a PS 390 display structure, it should not be used with programs whose structures may be changed.
2. To enter optimization mode for a developed application program, place the command

   ```
   OPTIMIZE STRUCTURE;
   ```

   at the beginning of the program (or portion of program) to be optimized, and place the command

   ```
   END OPTIMIZE;
   ```

   at the end.
3. Optimization is not retroactive. The OPTIMIZE STRUCTURE command alone does not optimize any existing structures. On the other hand, structures created after the command is entered remain optimized even after END OPTIMIZE is entered, and even after legal changes are made to the structure.

4. The following changes may not be made to structures created or instanced during optimization mode:
    a. PREFIXes
    b. Redefinitions of data-definition commands (VECtor_list, CHARacters, LABELS, and polynomial and B-spline curves), regardless of whether or not the system is in optimization mode at the time of redefinition. Illegal changes to optimized structures have unpredictable effects on the display.
5. Among the types of structures for which optimization has an effect are INSTANCEs of multiple data-definition commands and BEGIN_S...END_S structures containing only data-definition commands.
6. Optimization has no effect on a reference to a data-definition command which precedes the data-definition command itself.
7. OPTIMIZE STRUCTURE, like the INITialize command, affects only those structures created at the port at which the command is entered.
8. An INITialize command automatically performs an END OPTIMIZE.

PATTERN

TYPE

MODELING — Primitives

FORMAT

```
name := PATtern i [AROUND_corners][MATCH/NOMATCH] LENgth r;
```

DESCRIPTION

Defines name to be a pattern. Patterns can be applied to existing vector lists (patterned and unpatterned) created by the WITH PATTERN, POLYNOMIAL, and BSPLINE commands. If curve commands are used, the [AROUND_corners] option must be used.

PARAMETERS

i — A series of up to 32 integers between 0 and 128 (delineated by spaces) indicating the relative lengths of alternating lines, spaces, lines, etc., in the pattern. The longer the series, the more complex the pattern of lines and spaces, which repeats every r units.

AROUND_corners — This indicates that patterning is to continue around each of the vectors in the vector list until the end of the list or a position vector is reached.

MATCH/NOMATCH — This indicates that the pattern length should be adjusted to make the pattern exactly match the end points of the vector or series of vectors being patterned. The default is MATCH.

r — The length over which i is defined and repeated.

EXAMPLE

Refer to Helpful Hint 10 in Section *TT2*.

TYPE

MODELING — Primitives

FORMAT

```
PATTERN name1 WITH pattern;
```

DESCRIPTION

An immediate-action command which applies a pattern to a vector list (name1).

PARAMETERS

pattern — The pattern to be applied to name1. The pattern can be defined as either of the following.

name — A pattern created by the name := PATtern command, or

i [AROUND_corners] [MATCH/NOMATCH] LENgth r

where:

i is a series of up to 32 integers between 0 and 128 delineated by spaces indicating the relative lengths of alternating lines, spaces, lines, etc., in the pattern. The longer the series, the more complex the pattern of lines and spaces, which repeats every r units.

AROUND_corners indicates that patterning is to continue around each of the vectors in the vector list until the end of the list or a position vector is reached.

MATCH/NOMATCH indicates that the pattern length should be adjusted to make the pattern exactly match the end points of the vector or series of vectors being patterned. The default is MATCH.

r is the length over which **i** is defined and repeated.

POLYGON

TYPE

MODELING — Primitives

FORMAT

```
name := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [COPLANAR]
        POLYGon vertex ... vertex;
```

DESCRIPTION

Allows you to define primitives as solids and surfaces. For a detailed explanation of defining and interacting with polygons, consult Section *GT13 Polygonal Rendering*.

PARAMETERS

WITH ATTRIBUTES — An option that assigns the attributes defined by name1 for all polygons until superseded by another WITH ATTRIBUTES clause.

WITH OUTLINE — An option that specifies the color of the outline to be drawn around polygon borders in enhanced-edge shaded images, or the color of polygon edges in hidden-line renderings.

COPLANAR — Declares that the specified polygon and the one immediately preceding it have the same plane equation.

vertex — A vertex is defined as follows:

```
[ S ] x,y,z [ N x,y,z ] [C h[,s[i]]]
```

where:

S — indicates that the edge drawn between the previous vertex and this one represents a soft edge of the polygon. If the S specifier is used for the first vertex in a polygon definition, the edge connecting the last vertex with the first is soft.

N — Indicates a normal to the surface with each vertex of the polygon. Normals are used only in smooth-shaded renderings. Normals must be specified for all vertices of a polygon or for none of them. If no normals are given for a polygon, they are defaulted to the same as the plane equation for the polygon.

x,y,z — are coordinates in a left-handed Cartesian system.

C — indicates a color to be assigned to the vertex. During shaded operations, this color is interpolated across the polygon to the other vertices. Color must be specified for all vertices of a polygon or none of them.

h,s,i — are coordinates of the Hue-Saturation-Intensity color system.

Polygons may be solidly colored by specifying a color through the attributes command or the colors may be assigned to the vertices by giving a color with each vertex specified. The color is specified by giving first the vertex and then the color (h, s, i). If just the hue and saturation are given, the intensity will default to 1. If just the hue value is given, the saturation and intensity will default to 1. If no vertex colors are given, the vertex colors will default to those specified in the ATTRIBUTE clause.

Vertex colors must be specified for all vertices of a polygon or for none of them. However, as with normals, some polygons may have color at their vertices while others polygons do not have color at their vertices. This means that it is possible to have some objects in the picture color interpolated, while others are not.

Although color of polygon vertices is specified h, s, i, the colors are linearly interpolated across the vertices in the Red-Green-Blue color system. If colors are not interpolating the way you would like them to, add more vertices to the polygon, or break up large solid volumes into smaller sub-volumes and assign the desired colors to the new vertices in the object.

You can specify color for a polygon with both the ATTRIBUTES command and the color by vertex specification. A new input to the SHADINGENVIRONMENT function allows you to switch between attribute-defined color and vertex-defined color. Input <10> of SHADINGENVIRONMENT accepts a Boolean to determine how color will be specified. To use vertex colors rather than surface attributes, send TRUE to input <10> of SHADINGENVIRONMENT. To return to using the attributes specified in the ATTRIBUTE command, send FALSE to input <10> of SHADINGENVIRONMENT.

NOTES

1. A polygon declared to be coplanar must lie in the same plane as the previous polygon if correct renderings are to be obtained. The system does not check for this condition. Coplanar polygons may be defined without the COPLANAR specifier, unless outer and inner contours are being associated.
2. To use the COPLANAR specifier to define a hole, the vertices of the hole must be ordered in a counter-clockwise direction, while the vertices of the surrounding polygon must be ordered in a clockwise direction.
3. All members of a set of consecutive coplanar polygons are taken to have the same plane equation, that of the previous polygon not containing the coplanar option. If coplanar is specified for the first polygon in a node, it has no effect.
4. If the N (normal) specifier is specified for a vertex in a polygon, it must be specified for all vertices in that polygon. The same is true for the C (color at vertex) specifier.
5. If the S (soft) specifier is used for the first vertex in a polygon definition, the edge connecting the last vertex with the first is soft.
6. No more than 250 vertices per POLYGon may be specified.
7. The last defined vertex in the polygon is assumed to connect to the first defined vertex; that is, polygons are implicitly closed.
8. There is no syntactical limit for the number of POLYGon clauses in a group.
9. The ordering of vertices within each POLYGon has important consequences for rendering operations.

DISPLAY STRUCTURE NODE CREATED

POLYGon data node.

INPUTS FOR UPDATING NODE

None.

TYPE

MODELING — Primitives

FORMAT

```
name := POLYnomial[ORDER=i]
        [COEFFICIENTS=]   xi,    yi,   zi
                          xi-1,  yi-1, zi-1
                           .      .     .
                           .      .     .
                           .      .     .
                          x0,    y0,   z0
        CHORDS=q;
```

DESCRIPTION

Evaluates a parametric polynomial in the independent variable t over the interval [0,1]. This command allows the parametric description of many curve forms without the need to specify or transfer the coordinates of each constituent vector.

If the polynomial to be evaluated is called C, C is an ith-order parametric polynomial in t such that:

```
C(t) = [ x(t) y(t) z(t) ]
```

This polynomial may be expressed as the product of a vector (containing the various powers of t) and a coefficient matrix with three columns and i+1 rows:

$$C(t) = [t_i\ t_{i-1}\ \ldots\ t_0] \begin{bmatrix} x_i & y_i & z_i \\ x_{i-1} & y_{i-1} & z_{i-1} \\ \vdots & \vdots & \vdots \\ x_0 & y_0 & z_0 \end{bmatrix}$$

This coefficient matrix is what is specified in the polynomial command to represent the parametric polynomial C.

PARAMETERS

i — Optional specification of the order of the polynomial used to anticipate internal storage requirements.

xi, yi, zi — Coefficients of the polynomial.

q — The number of vectors to be created ($0 < q < 32768$).

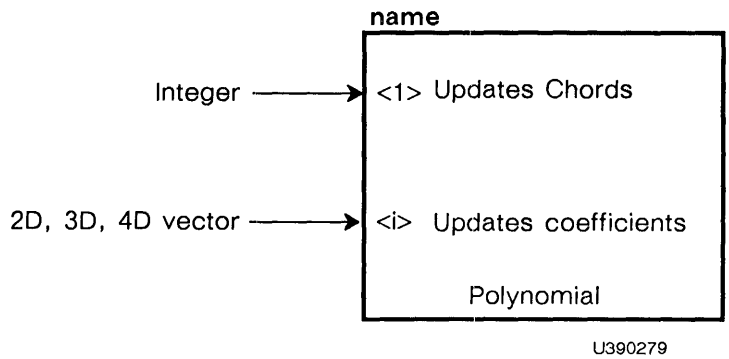
NOTES

1. The interval [0,1] over which the polynomial in t is to be evaluated is divided into q equal parts, so that C(t) is evaluated at t=0/q,1/q,2/q,...q/q. This causes the curve's constituent vectors generally not to be equal in length.
2. The polynomial's order is determined by the number of coefficient rows, and if the ORDER=i clause disagrees, it is ignored.

DISPLAY STRUCTURE NODE CREATED

Polynomial vector-list data node.

INPUTS FOR UPDATING NODE



NOTE ON INPUTS

Sending a 2D vector to a 3D polynomial node causes a default value of 0 to be used for Z. If a 4D vector is sent to a 3D polynomial or a 3D or 4D vector is sent to a 2D polynomial, the W or Z components are ignored.

PREFIX WITH

TYPE

STRUCTURE — Modifying

FORMAT

```
PREFIX name WITH operation_command;
```

DESCRIPTION

Prefixes a named data node (name) with an operation node.

PARAMETERS

name — A modeling primitive data node to be prefixed.

operation_command — Any command that creates an operation node.

NOTE

Any connections made to name will be applied to the added prefix and not to the modeling primitive (i.e. name now points to the new operation node which points to the node that was previously name).

DISPLAY STRUCTURE NODE CREATED

None. This is an immediate-action command which just modifies an existing data node.

EXAMPLE

```
A:= VECtor_list ...;
PREfix A WITH SCALE by .1;

{This will make A the name of a scaling node pointing at a now-unnamed
vector list.}
```

TYPE

MODELING — Primitives

FORMAT

```
name := RATIonal BSpline ORDER=k
                [OPEN/CLOSED] [NONPERIodic/PERIodic] [N=n]
                [VERTICES =]  x1,y1,[z1],w
                              x2,y2,[z2],w2
                               .  .   .   .
                               .  .   .   .
                               .  .   .   .
                              xn,yn,[zn],wn
                [KNOTS = t1,t2,...,tj]
                CHORDS =q;
```

DESCRIPTION

Evaluates a rational B-spline curve, allowing the parametric description of the curve form without the need to specify or transfer the coordinates of each constituent vector.

The rational B-spline curve C is defined as:

$$C(t) = \frac{\sum_{i=1}^{n} wipiNi,k(t)}{\sum_{i=1}^{n} wipiNi,k(t)}$$

where

pi — ith vertex of the B-spline's defining polygon

Ni,k — ith B-spline blending function of order k

wi — weighting factor associated with each vertex (different weights determine the shape of the curve).

The parameter t of the curve and blending functions is defined over a sequence of knot intervals t1,t2,...,tn+k. Different knot sequences define different types of B-splines.

Two common knot sequences are the uniform nonperiodic and uniform periodic knot sequences. A uniform nonperiodic B-spline is defined by the knot sequence:

$$tj = \begin{cases} 0 & (\text{for } j < k) \\ j-k & (\text{for } k < j < n) \\ n-k+1 & (\text{for } n < j < n+k) \end{cases}$$

A uniform periodic B-spline is defined by the knot sequence:

$$tj = j \quad (\text{for } j < n+k)$$

The blending functions can be defined recursively as

$$Ni,1(t) = 1 \text{ (if } ti < t < ti+1), \text{ 0 otherwise}$$

$$Ni,k(t) = \frac{(t-ti)Ni,k-1(t)}{ti+k-1-ti} + \frac{(ti+k-t)Ni+1,k-1(t)}{ti+k-ti+1}$$

The curve is evaluated at the points:

$$t = \frac{(1 - i)tk + itj - k + 1}{q}$$

for i=0,1,2,...,q.

PARAMETERS

k — The order of the curve (0 < k < 10).

n — The number of vertices (used to anticipate storage requirements).

x1,y1,z1,w1...xn,yn,zn,wn — The vertices and weighting factor of the defining polygon of the curve. The Z component is optional.

t1,t2,...,tj — User-specified knot sequence. Because closed B-splines are evaluated as open B-splines with duplicate vertices, the number of knots required is:

n+k	for open B-splines
n+k+1	for closed nonperiodic B-splines
n+2k-1	for closed periodic B-splines

The knots must also be nondecreasing.

q — The number of vectors to be created (0 < q < 32766).

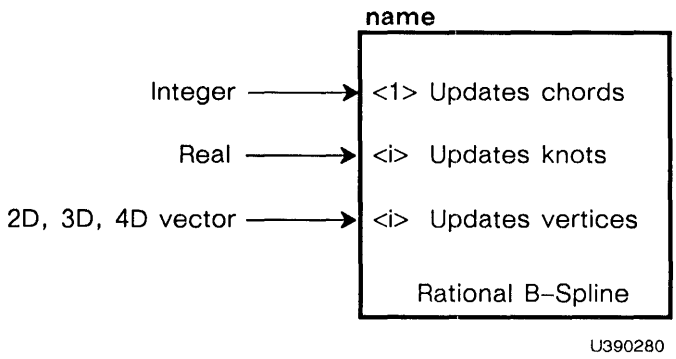
NOTES

1. OPEN or CLOSED is an option which describes the B-spline defining polygon. The default is OPEN. (Note that CLOSED merely describes the polygon, eliminating repetition of vertices. A full knot sequence, if specified, must be given.)
2. NONPERIODIC or PERIODIC is an option which specifies the default knot sequence. NONPERIODIC is the default for open B-splines; PERIODIC is the default for closed B-splines.
3. At least k vertices must be given, or the order k will be reduced accordingly.
4. If all the weights of a rational B-spline are the same, the curve is identical to the B-spline without the weights.

DISPLAY STRUCTURE NODE CREATED

B-spline vector-list data node.

INPUTS FOR UPDATING NODE



NOTE ON INPUTS

When a 2D vector is sent to a 3D rational B-spline, the default for Z is 0 and for W is 1. The third component of 3D and 4D vectors is used as W in 2D rational B-splines.

EXAMPLE

A third-order rational B-spline with defining polygon P1, P2, P3 defines a conic arc:

- the arc is parabolic if w1=w2=w3
- the arc is elliptic if w1=w3>w2
- the arc is hyperbolic if w1=w3<w2

TYPE

MODELING — Primitives

FORMAT

```
name := RATional POLYnomial[ORDER=i]
        [COEFFICIENTS=]  xi,   yi,   zi,   wi
                       xi-1, yi-1, zi-1, wi-1
                         .     .     .     .
                         .     .     .     .
                         .     .     .     .
                        x0,   y0,   z0,   w0
        CHORDS=q;
```

DESCRIPTION

Evaluates a rational parametric polynomial in the independent variable t over the interval [0,1]. This command allows the parametric description of many curve forms without having to specify or transfer the coordinates of each constituent vector.

If the polynomial to be evaluated is called C, C is an ith-order rational parametric polynomial in t such that:

$$C(t) = \left[\frac{x(t) \quad y(t) \quad z(t)}{w(t) \quad w(t) \quad w(t)} \right]$$

This polynomial may be expressed as the product of a vector (containing the various powers of t) and a coefficient matrix with four columns and i+1 rows:

$$C(t) = [ti \quad ti-1 \quad \ldots \quad t0] \begin{bmatrix} xi & yi & zi & wi \\ xi-1 & yi-1 & zi-1 & wi-1 \\ \vdots & \vdots & \vdots & \vdots \\ x0 & y0 & z0 & w0 \end{bmatrix}$$

This coefficient matrix is what is specified in the polynomial command to represent the rational parametric polynomial C.

PARAMETERS

i — Optional specification of the order of the polynomial used to anticipate internal storage requirements.

xi, yi, zi, wi — Coefficients of the polynomial.

q — The number of vectors to be created (0<q<32768).

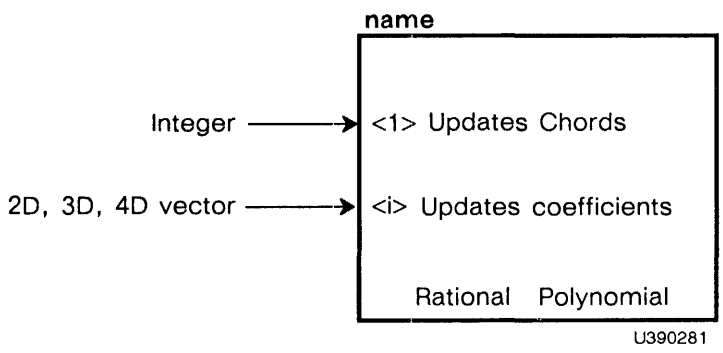
NOTES

1. The interval [0,1] over which the polynomial in t is to be evaluated, is divided into q equal parts, so that C(t) is evaluated at t=0/q,1/q,2/q,...q/q.
2. Note that the curve's constituent vectors are not generally equal in length.
3. The polynomial's order is determined by the number of coefficient rows, and if the ORDER=i clause disagrees, it is ignored.

DISPLAY STRUCTURE NODE CREATED

Rational-polynomial vector-list data node.

INPUTS FOR UPDATING NODE



NOTE ON INPUTS

Sending a 2D vector to a 3D polynomial node causes a default value of 0 to be used for Z and 1 for W. If a 4D vector is sent to a 3D polynomial or a 3D or 4D vector is sent to a 2D polynomial, the W or Z and W components are ignored. The third component of 3D and 4D vectors is used as W in a 2D rational polynomial.

EXAMPLES

```
Circle:= BEGIN_Structure

          RATional POLYnomial
          -2,  0,  0,  2
          -2, -2,  0, -2
           0,  1,  0,  1
         CHORDS = 25;

         RATional POLYnomial
          -2,  0,  0,  2
           2, -2,  0, -2
           0,  1,  0,  1
         CHORDS = 25;

         END_Structure;

{This will create right and left semicircles of radius 1.}
```

RAWBLOCK

TYPE

ADVANCED PROGRAMMING — Memory Allocation

FORMAT

```
name := RAWBLOCK i;
```

DESCRIPTION

Used to allocate memory that can be directly managed by a user-written function or by the physical I/O capabilities of the Parallel or Ethernet Interfaces.

PARAMETER

i — bytes available for use.

NOTES

1. The command carves a contiguous block of memory such that there are "i" bytes available for use.
2. The block looks like an operation node to the ACP. The descendent alpha points to the next long word in the block. What the ACP expects in this word is the .datum pointer of the alpha block. (The datum pointer points to the first structure to be traversed by the ACP. This is the address in memory where the data associated with a named entity is located.)
3. To use this block, the interface or user-written function fills in the appropriate structure following the .datum pointer. When this is complete, it changes the .datum pointer to the proper value and points to the beginning of the data. After the ACP examines this structure, it displays the newly-defined data. (Use the ACPPROOF procedure to change the .datum pointer with a user-written function.)
4. More than one data structure at a time can exist in a RAWBLOCK. It is up to the user to manage all data and pointers in RAWBLOCK.

5. A RAWBLOCK may be displayed or deleted like any other named data structure in the PS 390. When a RAWBLOCK is returned to the free storage pool, the PS 390 firmware recognizes that it is a RAWBLOCK and does not delete any of the data structures linked to RAWBLOCK.

DISPLAY STRUCTURE NODE CREATED

Rawblock data node.

REBOOT

TYPE

GENERAL — Command Control and Status

FORMAT

```
name := REBOOT password;
```

DESCRIPTION

Causes the PS 390 to reboot just as if it had been powered up; that is, it starts the confidence tests beginning with "A."

PARAMETER

password — System password

NOTES

1. If a password has been set up, an incorrect password will give an error message. If no password has been setup, any character string will cause the PS 390 to reboot.
2. REBOOT may be used inside a BEGIN_Structure ... END_Structure or outside.

TYPE

GENERAL — Data Structuring and Display

FORMAT

```
REMove name;
```

DESCRIPTION

Stops the display of **name**, that is, removes **name** from the display list.

PARAMETER

name — Any structure name.

NOTE

Does not affect any structures in memory.

REMOVE FOLLOWER

TYPE

STRUCTURE — Modifying

FORMAT

```
REMove FOLLOWER of name;
```

DESCRIPTION

Removes a previously placed follower of **name** (see FOLLOW WITH command).

PARAMETER

name — Structure that was previously modified with a FOLLOW WITH command.

EXAMPLE

(Refer to the example given in the FOLLOW WITH command.)

```
REMove FOLLOWER of Shape.Rot;

{This command will restore the structure Shape to what it was
originally (i.e. before the FOLLOW WITH command was given.)}
```

TYPE

STRUCTURE — Modifying

FORMAT

```
REMove name1 FROM name2;
```

DESCRIPTION

Used to remove a named node (name1) from a named instance node (name2) in a display structure.

PARAMETERS

name1 — Node to be removed from instance node name2.

name2 — Instance node that will no longer point to name1.

DISPLAY STRUCTURE NODE CREATED

None. This is an immediate-action command which modifies an existing instance node.

EXAMPLE

```
Map:= INSTance Canada, South_America, United_States;

REMOVE South_America FROM Map;

{This makes the instance of Map point at Canada and United_States
only.}
```

REMOVE PREFIX

TYPE

STRUCTURE — Modifying

FORMAT

```
REMove PREfix of name;
```

DESCRIPTION

Removes a previously placed prefix (see PREFIX WITH command).

PARAMETER

name — Structure that was previously modified by a PREFIX WITH command.

NOTE

This immediate-action command restores name to what it was before being modified by a PREFIX WITH command.

EXAMPLE

```
A:= VECtor_list ...;
PREfix A WITH SCALE by .1;
REMove PREfix of A;

{This will remove the previously PREfixed SCALE node, and A will once
again be the name of the VECtor_list.}
```

RESERVE_WORKING_STORAGE

TYPE

GENERAL — Immediate Action

FORMAT

```
RESERVE_WORKING_STORAGE size;
```

DESCRIPTION

Reserves a contiguous block of mass memory for sectioning plane, hidden-line removal, and backface removal renderings of solid objects defined as polygons.

PARAMETER

size -- The number of bytes of mass memory that are reserved.

NOTES

1. Renderings and saved renderings reside in mass memory along with the rest of the display structure. The original polygon is also stored in mass memory.
2. Each polygon of a solid object with four vertices will require approximately 150 bytes of reserve working storage. Memory needs will vary from figure to figure dependent upon the complexity of the object, the operations to be performed, and the view.
3. After one reserve-working-storage request is made, subsequent requests do not add to the original memory block — they replace the original memory block.
4. If a contiguous block of memory cannot be allocated, no working storage is allocated and any previous storage is deallocated.
5. The best time to use RESERVE_WORKING_STORAGE is after booting, when large requests can be filled more easily. However, the command may be entered at any time.

6. Typically, 200,000 to 400,000 bytes of working storage should be reserved at the beginning of a session.
7. A previously allocated block of memory is released prior to filling the request for a new block. Thus, a request for a smaller working storage area can always be fulfilled. However, because the working storage must be a contiguous block of memory, even slight increases in the working storage size may not be satisfied.
8. If working storage is too small or has not been reserved, additional storage will be allocated, which may not be contiguous. Rendering will be performed but at a slower rate than if the working storage were a contiguous block.

TYPE

GENERAL — Command Control and Status

FORMAT

```
!RESET;
```

DESCRIPTION

The !RESET command is used to get out of unended BEGINs or BEGIN_Structures when a problem occurs. (Refer also to COMmand STATus.)

ROTATE

TYPE

MODELING — Transformations

FORMAT

```
name := ROTate in [axis] angle [APPLied to name1];
```

DESCRIPTION

Rotates a structure (**name1**). Creates a 3x3 rotation matrix which rotates the specified data (vector lists and/or characters) about the designated **axis**, relative to the world coordinate system's origin. When you look in the positive direction of a given axis, positive **angle** values cause counterclockwise rotations (following the left-hand rule).

PARAMETERS

axis — X, Y, or Z. If no axis is specified, the default is Z.

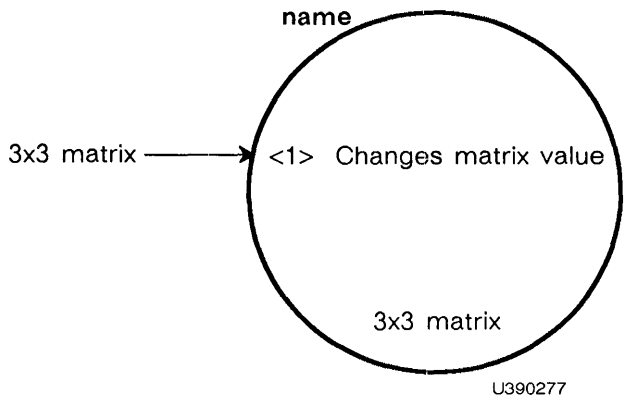
angle — Rotation angle in degrees (if no other units have been specified as default, and if no other units are explicitly specified in the ROTate command).

name1 — Structure to be rotated.

DISPLAY STRUCTURE NODE CREATED

3x3-matrix operation node.

INPUT FOR UPDATING NODE



NOTE ON INPUT

Any 3x3 matrix is legal (any rotation matrix, a scaling matrix, a compound 3x3 matrix, etc.).

ASSOCIATED FUNCTIONS

F:MATRIX3, F:XROTATE, F:YROTATE, F:ZROTATE, F:DXROTATE, F:DYROTATE, F:DZROTATE, F:SCALE, F:DSCALE

EXAMPLE

```
A:= ROTate in X 45 THEN B;
B:= VECtor_list ... ;
```

TYPE

MODELING — Transformations

FORMAT

```
name := SCALE by s [APPLied to name1];
name := SCALE by sx,sy[,sz] [APPLied to name1];
```

DESCRIPTION

Scales an object. Applies a uniform (s) or nonuniform (sx,sy,sz) 3x3 scaling matrix transformation to the specified data (vector lists and/or characters).

PARAMETERS

s — Uniform scaling factor (same along all axes).

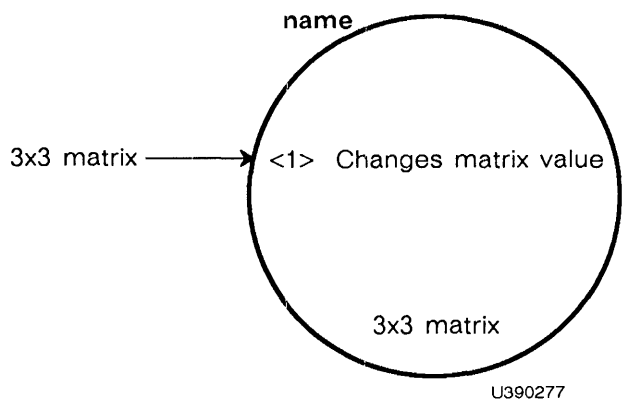
sx,sy,sz — Axial scaling factors. If **sz** is not specified, it is assumed to be 1 (no Z-scaling).

name1 — Object to be scaled.

DISPLAY STRUCTURE NODE CREATED

3x3-matrix operation node.

INPUT FOR UPDATING NODE



NOTE ON INPUT

Any 3x3 matrix is legal (another scaling matrix, a rotation matrix, etc.).

ASSOCIATED FUNCTIONS

F:MATRIX3, F:XROTATE, F:YROTATE, F:ZROTATE, F:DXROTATE, F:DYROTATE, F:DZROTATE, F:SCALE, F:DSCALE

EXAMPLE

```
A:= SCALE by 5,2,3 THEN B;
B:= VECtor_list ... ;
```

SECTIONING_PLANE

TYPE

RENDERING — Data Structuring

FORMAT

```
name := SECTioning_plane [APPLied to name1];
```

DESCRIPTION

Defines a sectioning plane, which is needed to produce a sectioned rendering of an object.

PARAMETER

name1 — Either a POLYGon command or an ancestor of a POLYGon command.

NOTES

1. Defining, displaying, and positioning a sectioning plane are the first steps in producing a sectioned rendering of an object. Hidden-line removal and backface removal do not require sectioning planes, but they can be used in conjunction with sectioned renderings.
2. The data which actually define a sectioning plane are contained in a POLYGon node; SECTioning_plane simply indicates that a given POLYGon represents a sectioning plane rather than an object to be rendered.
3. The sectioning plane is the plane in which a specified POLYGon lies. The polygon itself need not intersect the object to be sectioned, as long as some part of the plane does.
4. The sectioning plane is the plane containing the polygon defined by the first POLYGon clause of the first polygon node encountered by the display processor as it traverses the branch beneath a sectioning-plane node.

5. If the polygon node has more than one POLYGon, only the first polygon determines the sectioning plane. The other polygons have no effect on sectioning operations but are displayed along with the defining polygon. This can be put to good use in designing an indicator which shows the side of the plane at which sectioning will remove (or preserve) polygon data.
6. A node may be a descendant of a sectioning-plane node if and only if it may be a descendant of a rendering operation node. Refer to the Notes on the SOLID_rendering command for permitted and prohibited descendant nodes.
7. If objects are to be sectioned, matrix-transformation nodes may be placed above the sectioning-plane node when and only when they are also ancestors of the objects' SOLID_Rendering or SURFACE_Rendering node(s). Failure to observe this rule results in bad renderings.
8. No SOLID_rendering or SURFACE_rendering operation node, whether below or above the sectioning-plane node, may be an ancestor of a sectioning plane's defining POLYGon. These POLYGons are interpreted as objects to be rendered rather than as sectioning-plane definitions, and issues a "Sectioning plane not found" message when a sectioning attempt is made. Other nodes which do not represent matrix viewing transformations, such as SET RATE, may be placed either above or below the sectioning-plane node as needed.
9. Before an object can be sectioned, the sectioning-plane node must be part of a structure which is DISPlayed. If the plane's defining POLYGon is itself DISPlayed but its sectioning-plane node is not, no renderings can be created.

DISPLAY STRUCTURE NODE CREATED

SECtioning_plane operation node.

SELECT FILTER

TYPE

VIEWING — Appearance Attributes

FORMAT

```
name1 := Select Filter n THEN name2;
```

DESCRIPTION

This command selects one of the four line filters supported on the PS 390. The filters determine the type of aliased or antialiased line the system will draw.

PARAMETERS

n — 0,1,2,3 line filter selected where,

0. SIN(X)/X filter
1. narrow Gaussian (default)
2. wide Gaussian
3. jagged (no filter)

name2 — node to which filter is applied

DEFAULT

The default line filter is n=1, narrow Gaussian. Values outside the 0–3 range default to the narrow Gaussian with the following warning message:

```
W2045 ** Illegal filter selection, default filter 1 used
```

NOTES

1. The SIN(X)/X filter (filter 0) produces the sharpest, best quality lines and works well with images such as text characters that require fine detail. However, the SIN(X)/X filter only works with limited background colors; it works best with light background colors, such as gray. The SIN(X)/X filter produces more artifacts than the Gaussian filters when multiple lines overlap.

2. The default line filter is the narrow Gaussian filter (filter 1). The narrow Gaussian filter is the best general-purpose filter and produces good quality, sharp lines. It works with any background color and works well with detailed images such as those that contain radial lines.
3. The wide Gaussian filter (filter 2) creates wider lines with less definition. The wide Gaussian filter produces no artifacts and works well with primitives such as dots.
4. The jaggy filter (filter 3) produces unfiltered, aliased lines.

DISPLAY STRUCTURE NODE CREATED

SELECT FILTER Operation Node

TYPE

FUNCTION — Immediate Action

FORMAT

```
SEND option TO <n>name1;
```

DESCRIPTION

Sends a value to input **n** of function instance, node, or variable **name1**.

PARAMETERS

option — The value to be sent. This can be any of the following forms:

i — A real number (with or without decimal point).

FIX(i) — Designates i to be an integer value (without decimal point).

V2D(i,j) — 2D vector.

V3D(i,j,k) — 3D vector.

V4D(i,j,k,l) — 4D vector.

M2D(a11,a12 a21,a22) — 2x2 matrix.

M3D(a11,a12,a13 a21,a22,a23 a31,a32,a33) — 3x3 matrix

M4D(a11,a12,a13,a14 a21,a22,a23,a24 a31,a32,a33,a34 a41,a42,a43, a44) — 4x4 matrix

Boolean — TRUE or FALSE

'string' — A character string of one or more characters.

CHAR(m) — A single character whose decimal ASCII value is m.

P,L — Position or line.

VALUE(variable_name) — The value currently in variable_name, where variable_name is a previously declared PS 390 variable.

EXAMPLE

```
Timer:= F:CLCSECONDS;
SEND FIX(10) TO <1>Timer;

{This puts an integer 10 on input 1 of TIMER.}
```

SEND number*mode

TYPE

FUNCTION — Immediate Action

FORMAT

```
SEND number*mode TO <n>name1;
```

DESCRIPTION

Sends to a vector list or labels node to change a specified number of vectors from position vectors to line vectors, or to turn a specified number of labels on or off.

PARAMETERS

number — An integer specifying the number of vectors or labels.

mode — Either a **P** or **L**. For vector lists, P indicates a position vector and L indicates a line vector. For a labels block, P turns the label off, L turns it on.

n — An integer which identifies the first vector or label to receive the new specification.

name1 — The destination vector-list or labels node.

TYPE

FUNCTION — Immediate Action

FORMAT

```
SEND VL(name1) TO <i>name2;
```

DESCRIPTION

Overwrites or appends vectors in vector lists or labels in label blocks.

PARAMETERS

name1 — Name of vector list, character string, or label block to be sent.

name2 — Name of the destination vector-list or labels node.

i — An integer that specifies the first vector or first label to be replaced in name2 with vectors or labels in name1.

NOTES

1. The parameter i can be replaced with last or append.
2. If i exceeds the number of vectors or labels in name2, the command will be ignored.

SET BLINKING ON/OFF

TYPE

STRUCTURE — Attributes

FORMAT

```
name := SET BLINKing switch [APPLied to name1];
```

DESCRIPTION

This command turns blinking on and off. It affects all objects below the node created by the command in the display structure.

PARAMETERS

switch — Boolean value. TRUE indicates that blinking will occur in the displayed objects. FALSE turns blinking off.

name1 — The name of the structure that will be affected by the command.

DISPLAY STRUCTURE NODE CREATED

This command creates a set blinking on/off operation node in the display structure that determines whether blinking will occur in the objects positioned below it in the display structure.

INPUT FOR UPDATING NODE

The blinking on/off operation node can be modified by sending a Boolean value to input <1>.

SET BLINK RATE

TYPE

STRUCTURE — Attributes

FORMAT

```
name := SET BLINK RATE n [APPLied to name1];
```

DESCRIPTION

This command specifies the blinking rate in refresh cycles to be applied to all objects below the node created by the command in the display structure.

PARAMETERS

n — An integer designating the duration of the blink in refresh cycles. The blinking data will be on for **n** refreshes and off for **n** refreshes.

name1 — The name of the structure to which the blinking rate is applied.

DISPLAY STRUCTURE NODE CREATED

This command creates a set blinking rate operation node in the display structure that specifies the blinking rate for all objects below it.

INPUT FOR UPDATING NODE

The node can be modified by sending an integer to input <1> which will change the blinking rate.

SET CHARACTERS

TYPE

VIEWING — Appearance Attributes

FORMAT

```
name := SET CHARacters orientation [APPLied to name1];
```

DESCRIPTION

Sets the type of screen orientation you want for displayed character strings.

PARAMETERS

orientation — Three types of orientation may be set:

WORLD_oriented — Characters are transformed just like any part of the object containing them.

SCREEN_oriented — Characters are not affected by ROTate or SCALE transformations. Intensity and size of characters still vary with depth (Z-position).

SCREEN_oriented/FIXED — Characters are not affected by ROTate or SCALE transformations. They are always displayed with full size and intensity.

name1 — Structure affected by the SET CHARacters node.

DEFAULT

```
SET CHARacters WORLD_oriented;
```

DISPLAY STRUCTURE NODE CREATED

SET CHARacters operation node.

SET COLOR

TYPE

VIEWING — Appearance Attributes

FORMAT

```
name := SET COLOR hue,sat [APPLied to name1];
```

DESCRIPTION

Specifies the color of an object (name1).

PARAMETERS

hue — A real number greater than or equal to 0 and less than 360, where:
0 = pure blue, 120 = pure red, 240 = pure green, 360 = pure blue.

sat — A real number from 0 to 1 where:
0 = no saturation (white), and 1 = full saturation.

name1 — Structure to be colored.

DEFAULT

The default setting for both hue and sat is 0.

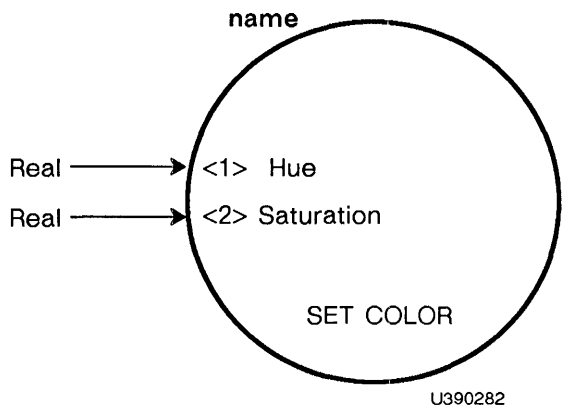
NOTE

Zero saturation in any hue is white.

DISPLAY STRUCTURE NODE CREATED

SET COLOR operation node.

INPUTS FOR UPDATING NODE



EXAMPLE

```
A:= SET COLOR 240,1 THEN B;
B:= Vector_list ......;

{If A is displayed, the vector list described by B will be displayed in
a pure green hue.}
```

SET CONDITIONAL_BIT

TYPE

STRUCTURE — Attributes

FORMAT

```
name := SET conditional_BIT n switch [APPLied to name1];
```

DESCRIPTION

Temporarily alters one of the 15 global conditional bits during the traversal of a branch of a display structure. These temporary settings may be tested further down the display structure, possibly allowing conditioned reference to other structures (see IF conditional_BIT command). When traversal of the branch is complete, the bits are restored to their previous values.

PARAMETERS

n — An integer from 0 to 14, corresponding to the conditional bit to be set ON or OFF by the command (Refer to Note 1 below).

switch — ON or OFF.

name — Structure to follow the conditional bit node.

DEFAULT

All 15 conditional bits are initially set to OFF.

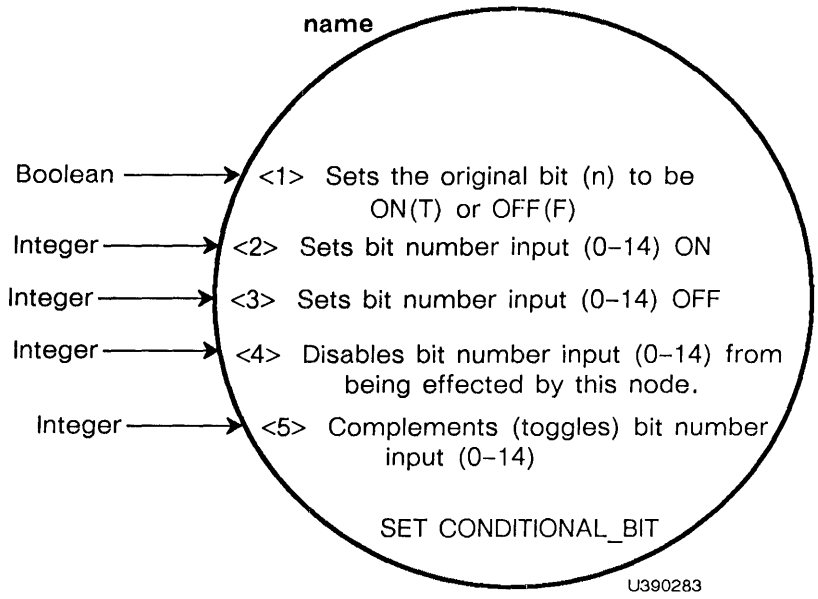
NOTES

1. Although only one conditional bit can be set ON or OFF by this command, a function network could be tied into this node to set any conditional bit ON or OFF.
2. Note that there is really only one bank of 15 conditional bits and that this command only changes the values of these bits temporarily, while name1 is being traversed. However, descendants of name1 could also be SET conditional_BIT nodes. These are saved and restored as part of the state of the machine during the traversal of different branches of the display structure.

DISPLAY STRUCTURE NODE CREATED

SET conditional_BIT operation node.

INPUTS FOR UPDATING NODE



EXAMPLE

```
A:= SET conditional_BIT 3 ON THEN B;
B:= IF conditional_BIT 3 is ON THEN C;
C:= IF conditional_BIT 6 is ON THEN D;
D:= VECtor_list ... ;

{A function network should be tied to A so that the state of any of the
conditional bits can be changed, not just the one that was initially
set ON or OFF.}
```

SET CONTRAST

TYPE

VIEWING — Appearance Attributes

FORMAT

```
name := SET CONTrast to c [APPLied to name1];
```

DESCRIPTION

Changes the contrast of the PS 390 display.

PARAMETERS

c — A number from 0 to 1 (0 = lowest contrast, 1= highest contrast).

name1 — Structure using this contrast setting.

DEFAULT

SET CONTrast to 1;

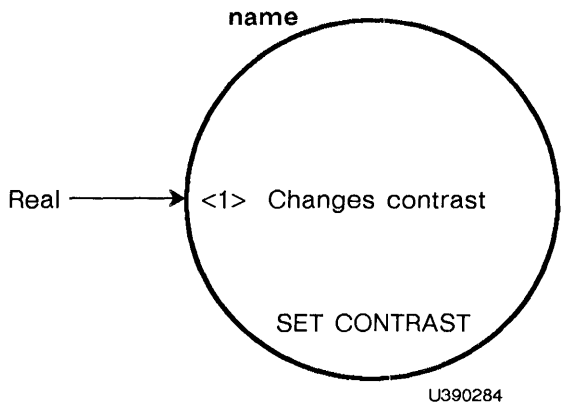
NOTES

1. Setting contrast to 1 provides the highest contrast and thus the greatest perception of depth cueing (all else being equal).
2. Although any real value from 0 to 1 is legal for c, c is mapped to one of four values (0.,.33,.67,1.).

DISPLAY STRUCTURE NODE CREATED

SET CONTrast operation node.

INPUT FOR UPDATING NODE



EXAMPLE

```
A:= SET CONTrast to 0 THEN B;
B:= VECtor_list ... ;

{This is a minimum contrast setting.}
```

SET DEPTH_CLIPPING

TYPE

VIEWING — Appearance Attributes

FORMAT

```
name := SET DEPTH_CLipping switch [APPLied to name1];
```

DESCRIPTION

Enables/disables Z-plane (depth) clipping.

PARAMETERS

switch — ON or OFF.

name1 — Structure affected.

DEFAULT

SET DEPTH_CLipping OFF;

NOTE

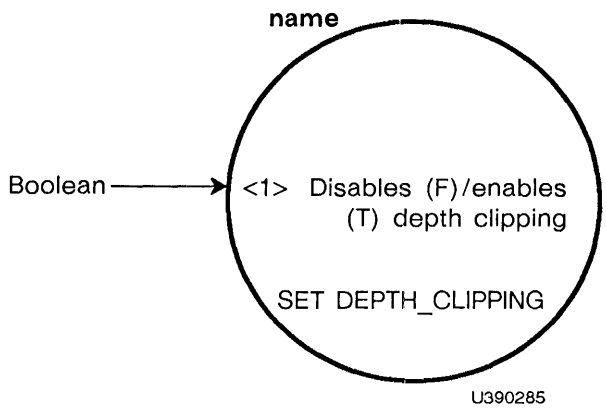
With depth clipping on (TRUE), data between the eye and the front clipping plane will be clipped, data between the front clipping plane and back clipping plane will appear with an intensity gradient, and data behind the back clipping plane will be clipped.

With depth clipping off (FALSE), data between the eye and front clipping plane will appear at full intensity, data between the front clipping plane and back clipping plane will appear with an intensity gradient, and data behind the back clipping plane will appear at minimum intensity.

DISPLAY STRUCTURE NODE CREATED

SET DEPTH_CLipping operation node.

INPUT FOR UPDATING NODE



EXAMPLE

```
A:= SET DEPTH_CLipping ON THEN B;
B:= ... ;

{This enables Z clipping.}
```

SET DISPLAYS

TYPE

VIEWING — Appearance Attributes

FORMAT

```
name := SET DISPlays ALL switch [APPLied to name1]:
name := SET DISPlay n[,m...] switch [APPLied to name 1];
```

DESCRIPTION

Sets the scope which receives display information to on/off.

PARAMETERS

switch — ON or OFF

n[,m...] — 0,1,2,3. Numeric designation for PS 390 Scope.

name1 — structure to be displayed

NOTE

The PS 390 only supports Scope 0.

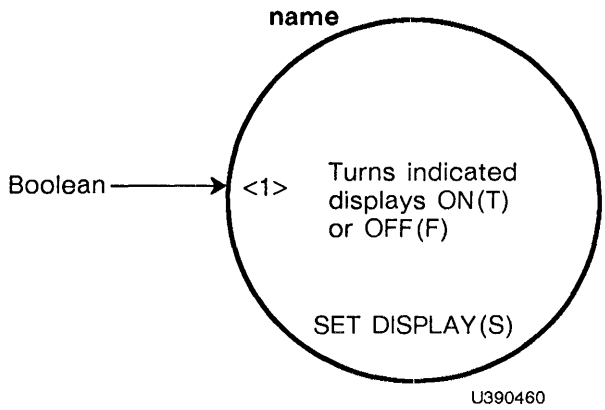
DEFAULT

```
SET DISPLAY 0 ON;
```

DISPLAY STRUCTURE NODE CREATED

SET DISPlay operation node

INPUT FOR UPDATING NODE



EXAMPLE

```
A:=SET DISPlay 0 ON THEN B;
B:=VECtor_list...;


{This channels B to be displayed on scope 0}
```

TYPE

VIEWING — Viewport Specification

FORMAT

```
name := SET INTENsity switch imin:imax [APPLied to name1];
```

DESCRIPTION

Specifies intensity variation for depth cueing, and may be used to override the intensity specification associated with the VIEWPORT command or previous SET INTENsity commands.

PARAMETERS

switch — Two settings may be specified: ON and OFF. The default setting is ON, which enables the effect of this node in the display structure. OFF disables the effect.

imin — A real number ranging from 0.0 to 1.0, imin represents the dimmest intensity setting.

imax — A real number ranging from 0.0 to 1.0, imax represents the brightest intensity setting.

name1 — Structure to be affected.

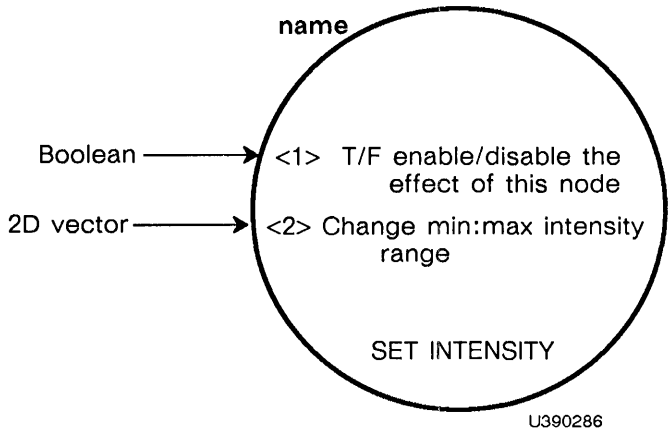
NOTE

The last SET INTENsity node that is ON in a display structure determines the intensity range.

DISPLAY STRUCTURE NODE CREATED

SET INTENsity operation node.

INPUTS FOR UPDATING NODE



EXAMPLE

Refer to Helpful Hint 15 in Section *TT2*.

SET LEVEL_OF_DETAIL

TYPE

STRUCTURE — Attributes

FORMAT

```
name := SET LEVel_of_detail to n [APPLied to name1];
```

DESCRIPTION

Temporarily alters a global level of detail value during the traversal of a specified branch of a display structure. These temporary settings may be tested further down the display structure, possibly allowing conditioned reference to other structures (see IF LEVel_of_detail command). When traversal of the branch is complete, the level of detail is restored to its original value.

PARAMETERS

n — An integer from 0 to 32767 indicating the level of detail value.

name — Structure to be affected by the level of detail.

DEFAULT

The level of detail is initially 0.

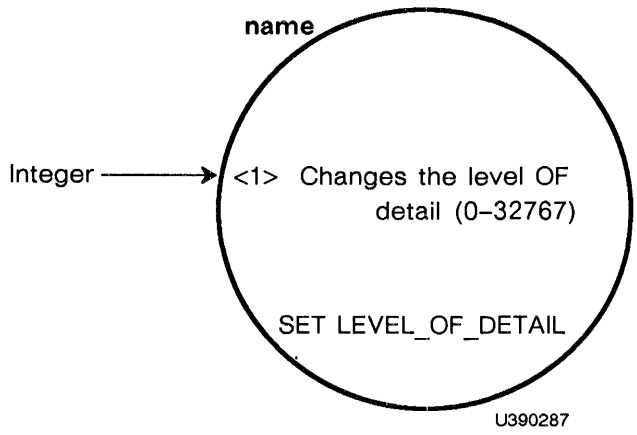
NOTE

There is really only one global level of detail value; this command only changes the value of the level of detail temporarily, while the name1 structure is being traversed.

DISPLAY STRUCTURE NODE CREATED

SET LEVel_of_detail operation node.

INPUT FOR UPDATING NODE



EXAMPLE

```
A:= SET LEVel_of_detail to 2 THEN B;
B:= IF LEVel_of_detail = 2 THEN C;
C:= ... ;

{A function network should be tied to A to change the level of detail
for conditional referencing of C.}
```

SET LINE_TEXTURE

TYPE

MODELING — Line Pattern

FORMAT

```
name := SET LINe_texture [AROUnd_corners] pattern [APPLied to name1];
```

DESCRIPTION

Specifies the line texture pattern to be used in drawing the vector lists that appear below the node created by this command. There are up to 127 hardware-generated line textures possible. The parameter pattern is an integer between 1 and 127. The desired line texture is indicated by the setting or clearing of the lower 7 bit positions in pattern when represented in binary. An individual pattern unit is 1.1 centimeters in length. Some of the more common patterns and their corresponding bit settings are shown below:

Pattern	Bit representation	Line Texture (repeated twice)	
127	1111111	----------	Solid
124	1111100	------ ------	Long Dashed
122	1111010	---- - ---- -	Long Short Dashed
106	1101010	-- - - -- - -	Long Short Short Dashed

PARAMETERS

AROUnd_corners — Boolean value used to set a flag to indicate if the specified line texture should continue from one vector to the next. If AROUnd_corners is TRUE, the line texture will continue from one vector to the next through the endpoint. If AROUnd_corners is FALSE, the line texture will start and stop at vector endpoints.

pattern — An integer between 1 and 127 that specifies the desired line texture. When pattern is less that 1 or greater than 127, solid lines are produced.

name1 — The name of the structure to which the line texture is applied.

DEFAULT

The default line texture is a solid line.

NOTES

1. Since 7 bit positions are used, it is not possible to create a symmetric pattern.
2. When line-texturing is applied to a vector, the vector that is specified is displayed as a textured, rather than solid line. If the line is smaller than the pattern length, then as much of the pattern that can be displayed with the vector is displayed. If the line is smaller than the smallest element of the pattern, then the line is displayed as solid.
3. The With Pattern and curve commands create multiple vectors in memory. To the line-texturing hardware, each vector in a pattern or curve is seen as an individual vector. Line-texturing a patterned line or curve is the same as line-texturing a number of small segments. Curves and patterns affect line-texturing only in that they tend to create short vectors that may be too short to be completely textured.

DISPLAY STRUCTURE NODE CREATED

This command creates a line texture operation node with line texture to be applied to all vectors below in the display structure hierachy. Sending a Boolean value to input <1> of the node turns the continuous texture feature on or off. Sending an integer value to the node changes the pattern.

EXAMPLE

Refer to Helpful Hint 10 in Section *TT2*.

TYPE

MODELING — Picking Attributes

FORMAT

```
name := SET PICKing switch [APPLied to name1];
```

DESCRIPTION

Enables or disables picking for a specified structure.

PARAMETERS

switch — ON or OFF for enabling or disabling picking.

name1 — Structure to be affected.

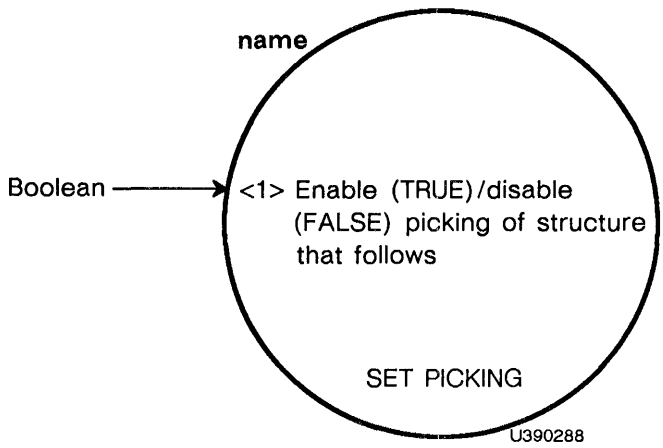
NOTES

1. For picking to be reported, there must also be a SET PICKing IDentifier node in the structure to be pickable.
2. Refer also to SET PICKing LOCation and SET PICKing IDentifier.

DISPLAY STRUCTURE NODE CREATED

SET PICKing operation node (information to enable/disable hardware picking).

INPUT FOR UPDATING NODE



EXAMPLE

```
A:= SET PICKing OFF THEN B;
B:= ... ;

{A function network should be tied to A to SET PICKing ON when needed
in order to make structure B pickable.}
```

SET PICKING IDENTIFIER

TYPE

MODELING — Picking Attributes

FORMAT

```
name := SET PICKing IDentifier = id_name [APPLied to name1];
```

DESCRIPTION

Specifies textual information that will be reported back if a pick occurs further down the structure name1. Nested pick identifier names are all reported, separated by commas.

PARAMETERS

id_name — Text that will be reported if a pick occurs anywhere within the structure name1. This must be a legal PS 390 name.

name1 — Structure to which the pick ID applies.

NOTES

1. At least one pick ID must precede any pickable entity for picking to be reported.
2. id_name cannot be updated by a function network.

DISPLAY STRUCTURE NODE CREATED

SET PICKing IDentifier operation node.

EXAMPLE

```
A:= SET PICKing OFF THEN B;
B:= SET PICKing IDentifier = Structure_C THEN C;
C:= VECtor_list ... ;

{If a vector in C is picked, the ID name reported in the pick list will
be Structure_C.}
```

SET PICKING LOCATION

TYPE

MODELING — Picking Attributes

FORMAT

```
name := SET PICKing LOCation = x,y size_x,size_y;
```

DESCRIPTION

Specifies a rectangular picking area at (x,y) within the current viewport. The rectangle is bounded by (x > size_x) and (y > size_y).

If an appropriate picking network is set up and a pick-sensitive vector list (vectors or dots) is drawn within the pick location, it will be reported as picked.

PARAMETERS

x,y — The center of the pick location.

size_x,size_y — Offsets from the x,y center specifying the bounds of the picking rectangle (the rectangle bounds must be within the range) 0–1.

DEFAULT

A default pick location is set up in the configuration file that is loaded when the system is booted. The **x,y** center is tied to the position of the data tablet stylus, and **size_x,size_y** are both set to .01, (i.e., a box whose dimensions are .02 on each side).

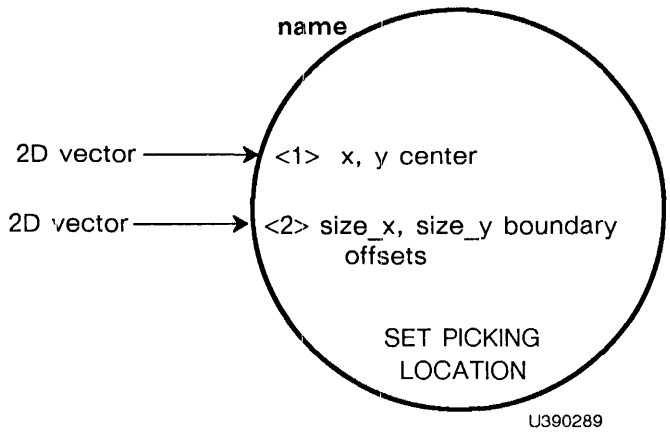
NOTES

1. In most applications, the picking location needs to be moveable, so the x,y center is usually updated by a function network that specifies where the center should be.
2. The data tablet's x,y value is usually the source for specifying the pick location center.

DISPLAY STRUCTURE NODE CREATED

SET PICKing LOCation operation node (information for hardware picking).

INPUTS FOR UPDATING NODE



ASSOCIATED FUNCTION

F:PICK

EXAMPLE

```
PICK_LOCATION := SET PICKing LOCation = 0,0 .02,.02;

{This redefines the default picking area set up in the configuration
file, making the picking area twice as large as the default.}
```

SET PRIORITY

TYPE

GENERAL — Command and Control Status

FORMAT

```
Set Priority of name to i;
```

DESCRIPTION

This command sets the execution priority of a function (**name**) to some integer (**i**) between 0 and 15. All functions instanced by the user and most functions instanced by the system at boot time have a default value of 8. Lowering a function's priority number raises its priority and causes it to run before any functions with a larger number. A typical use of this command is to give a function a priority number greater than 8 so it runs only when no other functions are running (i.e., functions at default priority 8). Assigning priority numbers less than 8 could be potentially very "dangerous," since their execution could lock up the system.

Since this command will affect the execution of other functions in a function network, careful consideration must be given to its use. E&S does not recommend the use of this procedure by anyone who does not have a complete understanding of functions and their interrelationships.

SET RATE

TYPE

STRUCTURE — Attributes

FORMAT

```
name := SET RATE phase_on phase_off [initial_state] [delay]
        [APPLied to name1];
```

DESCRIPTION

Temporarily alters two global duration values (**phase_on** and **phase_off**, in refresh frames) during the traversal of a specified branch of a display structure. These temporary settings may be tested further down the display structure, possibly allowing conditioned reference to other structures (see IF PHASE command). When traversal of the branch is complete, the durations are restored to their original values.

PARAMETERS

phase_on,phase_off — Integers designating the durations of the on and off phases, respectively, in refresh frames.

initial_state — ON or OFF, indicating the initial phase.

delay — Integer designating the number of refresh frames in the initial_state.

name1 — Structure to follow the SET RATE command.

DEFAULT

The default phase is OFF and never changes unless a SET RATE node is encountered.

NOTES

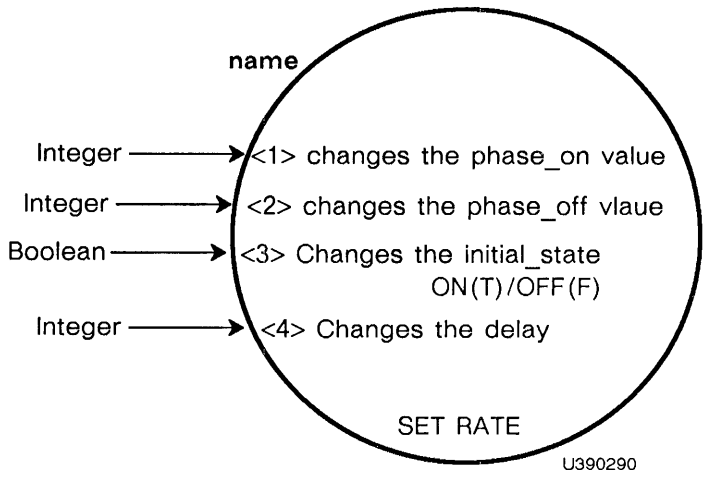
1. This structure attribute is useful for controlling blinking, the alternating display of two structures, the alternating display of a single structure in two different views (stereo), etc.

2. Note that there are only two rate values (phase_on, phase_off) and that this command only changes those values for the structure(s) that follow.

DISPLAY STRUCTURE NODE CREATED

SET RATE operation node.

INPUTS FOR UPDATING NODE



EXAMPLE

```
A:= BEGIN_Structure
rate:=  SET RATE 10 100;
        IF PHASE is ON THEN B;
    END_Structure;

B:= VECtor_list ... ;

{If A is DISPlayed, then vector list B will be displayed for 10 frames
and not displayed for 100 frames repetitively.}
```

SET RATE EXTERNAL

TYPE

STRUCTURE — Attributes

FORMAT

```
name := SET RATE EXTernal [APPLied to name1];
```

DESCRIPTION

Sets up a structure that can be used to alter the PHASE attribute via an external source, such as a function network or a message from the host computer. This PHASE attribute can be tested further down within the display structure, allowing conditional references to other structures (see IF PHASE command). See also the SET RATE command which alters the PHASE attribute based on refresh cycles.

PARAMETER

name1 — Structure to follow the SET RATE EXTernal command.

DEFAULT

The default phase is ON when a SET RATE EXTernal node is encountered.

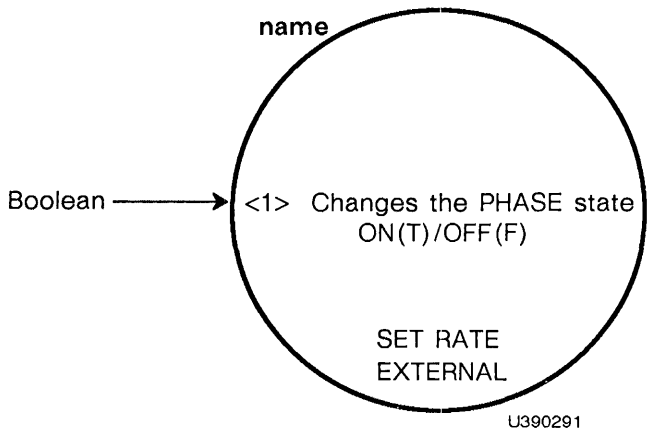
NOTES

1. The PHASE attribute is changed by sending a Boolean value to input 1 of SET RATE EXTernal node.
2. See also notes for SET RATE command.

DISPLAY STRUCTURE NODE CREATED

SET RATE EXTernal operation node.

INPUT FOR UPDATING NODE



EXAMPLE

```
A:= BEGIN_Structure
Rate:=  SET RATE EXTernal;
        IF PHASE is ON THEN B;
    END_Structure;

B:= VECtor_list ... ;

{A function network should be connected to A.Rate to set the PHASE ON
and OFF in order to conditionally display vector list B.}
```

TYPE

FUNCTION — Immediate Action

FORMAT

```
SETUP CNESS queue_type <i>name;
```

DESCRIPTION

Allows you to specify whether or not an input queue to a function instance is to be a constant queue.

PARAMETERS

queue_type — TRUE sets the queue type to constant, FALSE sets it to active.

name — Most intrinsic function names, except those listed in the notes.

NOTES

1. This feature should only be used when a function is first instanced. Input queues should not be changed between active and constant after the function has started processing data.
2. The SETUP CNESS command can be used for all intrinsic functions except the following.
   - F:BOOLEAN_CHOOSE
   - F:CI(n)
   - F:CLCSECONDS
   - F:CLFRAMES
   - F:CLTICKS
   - F:GATHER_GENFCN
   - F:INPUTS_CHOOSE(n)
   - F:K2ANSI

- F:LINEEDITOR
- F:LIST
- F:PICK
- F:RASTER
- F:TEDUP
- F:VT10

3. Functions which specify their queue characteristics by their name, e.g., F:ADDC, will continue to be instanced with their default active and constant queues.

SETUP INTERFACE

TYPE

GENERAL — Command and Control Status

FORMAT

```
SETUP INTERFACE portn/option=<n>;
```

DESCRIPTION

This command can be used to change any of the default values. These new values must be within the acceptable range of values for data characteristics.

PARAMETERS

portn — the port being reconfigured

option — the name of the value being changed

<n> — a legal parameter that is specific to the option

NOTES

1. In using this command, the port names are as follows:
   - Port 1 is designated port 10
   - Port 3 is designated port 30
   - Port 4 is designated port 40
   - Port 5 is designated port 50
2. The PS 390 does not have to be in Configure mode for this command.

EXAMPLE

```
SETUP INTERFACE port10/SPEED=300;
```

In this example, port 10 is the port being reconfigured, SPEED refers to the baud rate, and 300 is a legal speed for the communications interface. The effective baud rate for port10 is changed to 300.

Refer to Helpful Hint 14 in Section *TT2*.

SETUP PASSWORD

TYPE

GENERAL — Command and Control Status

FORMAT

```
SETUP PASSWORD password;
```

DESCRIPTION

This command allows you to establish and modify the password required to enter the Configure mode. This command can be included in the SITE.DAT file, or may be set up at any time. This command can only be entered while in Configure mode.

PARAMETER

password — the established string

SHOW INTERFACE

TYPE

GENERAL — Command and Control Status

FORMAT

```
SHOW INTERFACE <name>;
```

DESCRIPTION

This command can be used to check the values of a given port.

The menu available with the SHOW INTERFACE command when no parameter is given lists only those parameters that are relevant to the interface. For example, in synchronous mode, the X_ON/X_OFF parameter would not be listed.

PARAMETER

<name> — the port being checked

NOTES

1. In using this command, the port names are as follows:
    - Port 1 is designated port 10
    - Port 3 is designated port 30
    - Port 4 is designated port 40
    - Port 5 is designated port 50
2. The PS 390 does not have to be in Configure mode for this command.

SOLID_RENDERING

TYPE

MODELING — Data Structuring

FORMAT

```
name := SOLID_rendering [APPLied to name1];
```

DESCRIPTION

Declares a polygon object to be a solid and marks the object so that rendering operations can be performed on it. This command creates a rendering node.

PARAMETER

name1 — Either a POLYGon node or an ancestor of one or more POLYGon nodes.

NOTES

1. If non-POLYGon data nodes (VECtor_list, CHARacters, LABELS, POLYnomial, and BSPLINE) are included in name1, these data objects are displayed along with the polygon objects prior to rendering but are omitted from renderings. The rendering operations have no effect on these data nodes. However, special vector lists output from F:XFORMDATA used to display spheres and lines in the static viewport can be used and will be displayed if rendered.
2. IF and SET Conditional_BIT, IF and SET LEVel_of_detail, INCRement LEVel_of_detail, DECrement LEVel_of_detail, IF PHASE, SET RATE, SET RATE EXTernal, SET DEPTH_CLipping, and BEGIN_Structure ... END_Structure may be placed between a rendering node and its data. A rendering takes into account any effects of these nodes at the time the request is made; for example, if IF PHASE and SET RATE are being used to blink an object and that object is "off" at the moment the request is made, the object is excluded from the rendering.

   The nodes in the above paragraph may also be placed above the rendering node.

3. The transformations ROTate, TRANslate, SCALE, Matrix_2X2, Matrix_3X3, Matrix_4X3, and LOOK may be placed between a rendering node and its data node(s). However, these nodes should be used with caution, since, like the operation nodes mentioned above, their effects will be incorporated into renderings, and precision problems may result.

   Since most vertices in an object usually belong to more than one polygon, each vertex must be defined with the same numerical value in each of its polygons; otherwise, precision discrepancies may cause inaccurate renderings. The transformation nodes mentioned above may also be placed above the rendering node.

4. The five nodes WINDOW, VIEWport, EYE, Field_Of_View, and Matrix_4X4 should not, in general, be made descendants of a rendering node. Like other transformations, these five are incorporated into the output data from a rendering operation. However, this rendered data is generally displayed within a framework that already includes global 4x4-matrix transformations of its own. Including these transformations as part of the rendering, then, usually has the net effect of applying an unwanted double-WINDOW (double-VIEWport, etc.) to the rendered object.

5. SOLID_rendering, SURFACE_rendering, and SECTioning_plane may not be descendants of a rendering node, especially if multiple-instanced rendering nodes are involved. If this rule is not observed, bad renderings or a system crash may result. The system does not check for this condition.

6. Other nodes, including character transformations and the SET nodes (SET RATE, SET COLOR) not mentioned above, are ignored by rendering operations. Data nodes other than POLYGon are also ignored.

7. Before an object can be rendered, its rendering node must be part of a structure which is DISPlayed. If the object itself is DISPlayed but its rendering node is not, no renderings can be created.

8. Any input to input <1> of a rendering node causes an output. Inputs sent to input <2> will not cause an output to be sent. If output <1> has not been connected, and an integer, string, or Boolean is sent to input <1>, a message will appear on the screen upon successful completion of the rendering operation. An error message will appear if the rendering was not completed.

9. Input <3> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets the vectors as "moves" and "draws" for raster-line rendering.

10. Input <4> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets each vector as an x,y,z spherical primitive.

11. Input <5> of the rendering node accepts the name of the original vector list (sent to F:XFORMDATA with its output <1> sent to input <4> of the rendering node) to enable accurate scaling for rendering raster lines and spheres.

12. Toggling between the current rendering and the original object (sending a fix(0) to input <1> of the SOLID_rendering or SURFACE_rendering node) works only after requesting backface pictures, sectioned pictures, or cross-sectioned pictures.

13. Sending a fix(7) to input <1> of the SOLID_rendering or SURFACE_rendering node produces a type of Phong shading. Phong shading is made by interpolating the surface normal between vertices of the polygon and then calculating the correct lighting at each pixel. This is the highest quality of smooth shading currently supported.

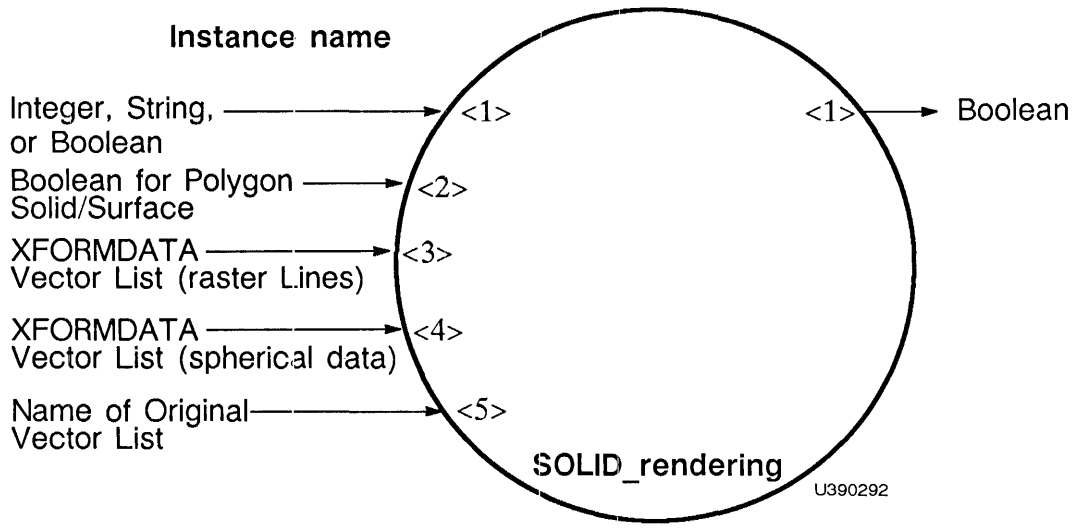
14. Sending a fix(8) to input <1> of the SOLID_rendering or SURFACE_rendering node will produce a type of Gouraud shading. Gouraud shading is made by calculating the correct lighting at the vertices of the polygon only and interpolating the intensity across the polygon to produce a smooth-shaded picture. An image produced with Gouraud shading will not be the quality of an image produced with Phong shading, but the Gouraud-shaded image will be produced at a faster rate. The user must supply normals at each of the polygons for the object to be smooth-shaded.

15. Sending data to a non-existent rendering node input will cause the system to crash.

DISPLAY STRUCTURE NODE CREATED

Rendering operation node.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

Input <1>

0: Toggles between the current rendering and the original object in the dynamic viewport.

1: Creates and displays a cross-section of an object (solid only) defined by the sectioning plane in the dynamic viewport.

2: Creates and displays a sectioned rendering in the dynamic viewport.

3: Creates and displays a rendering using backface removal (solid only) in the dynamic viewport.

4: Creates and displays a rendering using hidden-line removal in the static viewport.

5: Generates a wash-shaded image in the static viewport.

6: Generates a flat-shaded image in the static viewport.

7: Generates a Phong-shaded image in the static viewport.

8: Generates a Gouraud-shaded image in the static viewport.

String: Causes the current rendering to be saved under the name given in the string (dynamic viewport only).

False: Sets the original view. The original descendent structure of the rendering operation node is displayed.

True: Sets the rendered view. The rendered view of the original descendent structure of the operation rendering node.

Input <2>

True: Declares the object to be a solid.

False: Declares the object to be a surface.

Input <3>

Accepts a transformed vector list from output <1> of F:XFORMDATA to define raster lines.

Input <4>

Accepts a transformed vector list from output <1> of F:XFORMDATA to define spherical centers.

Input <5>

Accepts the original vector list to enable accurate spherical scaling.

Output <1>

True: Rendering is displayed.

False: Rendering is not displayed.

TYPE

VIEWING — Appearance Attributes

FORMAT

```
name := STANdard FONT [APPLied to name1];
```

DESCRIPTION

Establishes the standard PS 390 95-character font as the working font.

PARAMETER

name1 — Structure to use the standard font.

DEFAULT

If no other font is specified, the standard font is the default font.

NOTE

This command is necessary only if the standard font is to be used in a display structure that uses another font higher in the same structure.

DISPLAY STRUCTURE NODE CREATED

Character-font pointer node.

EXAMPLE

```
Slant := BEGIN_Font
         (character definitions)
         END_Font;

A := BEGIN_Structure
       character FONT Slant;
       CHARacters 'HERE';
       STANdard FONT;
       CHARacters 0,-2 'HERE';
     END_Structure;

DISPlay A;

{'HERE' at 0,0 will be in the Slant font 'HERE' at 0,-2 will be in the
standard font.}
```

TYPE

FUNCTION

FORMAT

```
STORE option IN name1;
```

DESCRIPTION

Sends a value to input <1> of function instance, node, or variable name1.

PARAMETERS

option — See SEND command.

name1 — function instance name, node name, or variable name to receive value on input <1>.

NOTE

This command is another way of doing a special case of the SEND command. It is synonymous with SEND option TO <1>name1;

EXAMPLE

```
Timer:= F:CLCSECONDS;
STORE FIX(10) IN Timer;

{This is equivalent to: SEND FIX(10) TO <1>Timer;}
```

SURFACE_RENDERING

TYPE

MODELING — Data Structuring

FORMAT

```
name := SURFACE_rendering [APPLied to name1];
```

DESCRIPTION

Declares a polygon object to be a surface and marks the object so that rendering operations can be performed on it. This command creates a rendering node.

PARAMETER

name1 — Either a POLYGon node or an ancestor of one or more POLYGon nodes.

NOTES

1. If non-POLYGon data nodes (such as VECtor_list, CHARacters, LABELS, POLYnomial, and BSPLINE) are included in name1, these data objects are displayed along with the polygon objects prior to rendering but are omitted from renderings. The rendering operations have no effect on these data nodes. However, special vector lists output from F:XFORMDATA used to display spheres and lines in the static viewport can be used and will be displayed if rendered.

2. IF and SET conditional_BIT, IF and SET LEVel_of_detail, INCRement LEVel_of_detail, DECrement LEVel_of_detail, IF PHASE, SET RATE, SET RATE EXTernal, SET DEPTH_CLipping, and BEGIN_Structure ... END_Structure may be placed between a rendering node and its data. A rendering takes into account any effects of these nodes at the time the request is made; for example, if IF PHASE and SET RATE are being used to blink an object and that object is "off" at the moment the request is made, the object is excluded from the rendering.

   The nodes in the above paragraph may also be placed above the rendering node.

3. The transformations ROTate, TRANslate, SCALE, Matrix_2X2, Matrix_3X3, Matrix_4X3, and LOOK may be placed between a rendering node and its data node(s). However, these nodes should be used with caution, since, like the operation nodes mentioned above, their effects will be incorporated into renderings, and precision problems may result.

   Since most vertices in an object usually belong to more than one polygon, each vertex must be defined with the same numerical value in each of its polygons; otherwise, precision discrepancies may cause inaccurate renderings. The transformation nodes mentioned above may also be placed above the rendering node.

4. The five nodes WINDOW, VIEWport, EYE, Field_Of_View, and Matrix_4X4 should not, in general, be made descendants of a rendering node. Like other transformations, these five are incorporated into the output data from a rendering operation. However, this rendered data is generally displayed within a framework that already includes global 4x4-matrix transformations of its own. Including these transformations as part of the rendering, then, usually has the net effect of applying an unwanted double-WINDOW (double-VIEWport, etc.) to the rendered object.

5. SOLID_rendering, SURFACE_rendering, and SECTioning_plane may not be descendants of a rendering node, especially if multiple-instanced rendering nodes are involved. If this rule is not observed, bad renderings or a system crash may result. The system does not check for this condition.

6. Other nodes, including character transformations and the SET nodes (SET RATE, SET COLOR) not mentioned above, are ignored by rendering operations. Data nodes other than POLYGon are also ignored.

7. Before an object can be rendered, its rendering node must be part of a structure which is DISPlayed. If the object itself is DISPlayed but its rendering node is not, no renderings can be created.

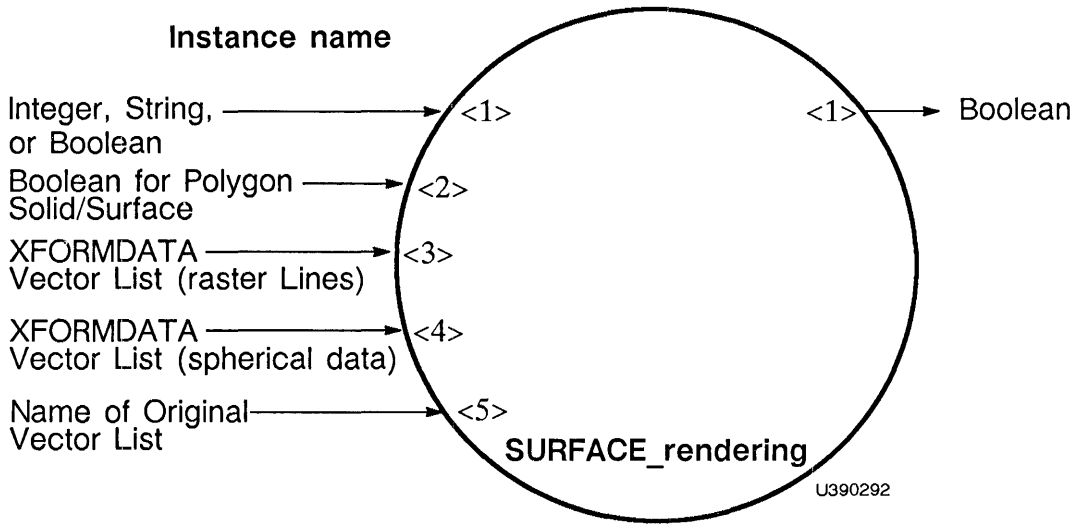
8. Any input to input <1> of a rendering node causes an output. Inputs sent to input <2> will not cause an output to be sent. If output <1> has not been connected, and an integer, string, or Boolean is sent to input <1>, a message will appear on the screen upon successful completion of the rendering operation. An error message will appear if the rendering was not completed.
9. Input of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets the vectors as "moves" and "draws" for raster-line rendering.
10. Input <4> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets each vector as an x,y,z spherical primitive.
11. Input <5> of the rendering node accepts the name of the original vector list (sent to F:XFORMDATA with its output <1> sent to input <4> of the rendering node) to enable accurate scaling for rendering lines and spheres.
12. Toggling between the current rendering and the original object (sending a fix(0) to input <1> of the SOLID_rendering or SURFACE_rendering node) works only after requesting backface pictures, sectioned pictures, or cross-sectioned pictures.
13. Sending a fix(7) to input <1> of the SOLID_rendering or SURFACE_rendering node produces a type of Phong shading. Phong shading is made by interpolating the surface normal between vertices of the polygon and then calculating the correct lighting at each pixel. This is the highest quality of smooth shading currently supported.
14. Sending a fix(8) to input <1> of the SOLID_rendering or SURFACE_rendering node will produce a type of Gouraud shading. Gouraud shading is made by calculating the correct lighting at the vertices of the polygon only and interpolating the intensity across the polygon to produce a smooth-shaded picture. An image produced with Gouraud shading will not be the quality of an image produced with Phong shading, but the Gouraud-shaded image will be produced at a faster rate. The user must supply normals at each of the polygons for the object to be smooth-shaded.

15. Sending data to a non-existent rendering node input will cause the system to crash.

DISPLAY STRUCTURE NODE CREATED

Rendering operation node.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

Input <1>

0: Toggles between the current rendering and the original object in the dynamic viewport.

1: Creates and displays a cross-section of an object defined by the sectioning plane (solid only) in the dynamic viewport.

2: Creates and displays a sectioned rendering in the dynamic viewport.

3: Creates and displays a rendering using backface removal (solid only) in the dynamic viewport.

4: Creates and displays a rendering using hidden-line removal in the static viewport.

5: Generates a wash-shaded image in the static viewport.

6: Generates a flat-shaded image in the static viewport.

7: Generates a Phong-shaded image in the static viewport.

8: Generates a Gouraud-shaded image in the static viewport.

String: Causes the current rendering to be saved under the name given in the string (dynamic viewport only).

False: Sets the original view. The original descendent structure of the rendering operation node is displayed.

True: Sets the rendered view. The rendered view of the original descendent structure of the rendering operation node is displayed.

Input <2>

True: Declares the object to be a solid.

False: Declares the object to be a surface.

Input <3>

Accepts a transformed vector list from output <1> of F:XFORMDATA to define raster lines.

Input <4>

Accepts a transformed vector list from output <1> of F:XFORMDATA to define a spherical center.

Input <5>

Accepts the original vector list to enable accurate spherical scaling.

Output <1>

True: Rendering is displayed.

False: Rendering is not displayed.

TEXT SIZE

TYPE

MODELING — Character Transformations

FORMAT

```
name := TEXT SIZE x [APPLied to name1];
```

DESCRIPTION

Creates a 2X2 uniform scale matrix which defines character size.

PARAMETERS

x — The size of the characters.

name1 — Structure containing the characters.

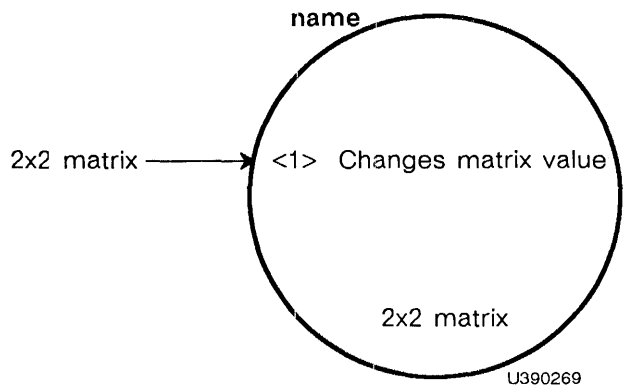
NOTES

1. The text size (x) is a multiple or fraction of the default character size, i.e. 1.
2. A TEXT SIZE node in a display structure overrides any previous character sizes that may have been created using the CHARacter SCAle or CHARacter SIZE commands. In other words, the TEXT SIZE scaling matrix is not concatenated into any other 2X2 matrix.
3. A TEXT SIZE node will also override CHARacter ROTate and Matrix_2X2 nodes.

DISPLAY STRUCTURE NODE CREATED

2x2-matrix operation node.

INPUT FOR UPDATING NODE



NOTE ON INPUT

Any 2x2 matrix is legal.

ASSOCIATED FUNCTIONS

F:MATRIX2, F:CSCALE

EXAMPLE

```
String := CHARacters 'This is only a test';
Scale := CHARacter SCAle 2 THEN String;
New_Scale := CHARacter SCAle 3 THEN Scale;
Change_Size := TEXT SIZE .5 THEN String;

{The Scale matrix creates characters twice the default size. The
New_Scale matrix is concatenated with the Scale matrix to create
characters six times the default size. The Change_Size matrix, however
is not concatenated, and so returns the characters to one half of the
default size.}
```

TYPE

MODELING — Transformations

FORMAT

```
name := TRANslate by tx,ty[,tz] [APPLied to name1];
```

DESCRIPTION

Translates an object by applying a translation vector to it.

PARAMETERS

tx,ty,tz — Distances to translate in each coordinate direction, in world coordinates.

name1 — Structure to be translated.

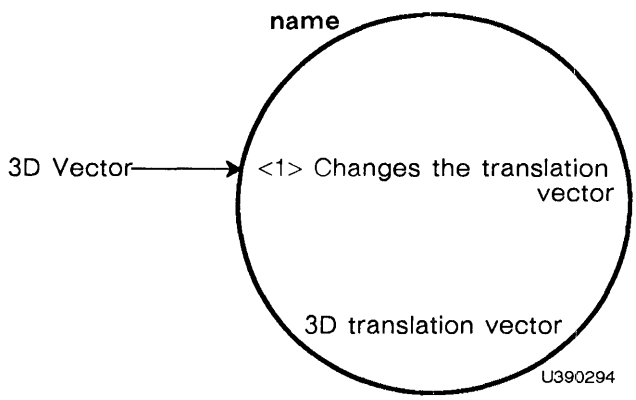
DEFAULT

tz is 0 if not specified.

DISPLAY STRUCTURE NODE CREATED

3D translation-vector operation node.

INPUT FOR UPDATING NODE



ASSOCIATED FUNCTIONS

F:XVECTOR, F:YVECTOR, F:ZVECTOR

EXAMPLE

```
A:= TRANslate by 5,7 THEN B;
B:= VECtor_list ... ;
```

VARIABLE

TYPE

FUNCTION — Data Structuring

FORMAT

```
VARiable name1[,name2 ... namen];
```

DESCRIPTION

Declares a storage place (or places) for any PS 390 function data type. A value can be stored in variable name1 either by SENDing (or STOREing) a value to input <1> of name1, or by CONNecting a function instance to input <1> of name1. The current value of variable name1 can be obtained by using either the F:FETCH function or the SEND VALUE (variable_name) option of the SEND command, where variable_ name in this case is name1.

PARAMETER

name1,name2... — Variable names.

EXAMPLE

```
VARiable Current_XY, X, Y, Z, Save;
```

TYPE

MODELING — Primitives

FORMAT

```
name := VECtor_list [options] [N=n] vectors;
```

DESCRIPTION

Defines an object by specifying the points comprising the geometry of the object and their connectivity (topology).

PARAMETERS

name — Any legal PS 390 name.

options — Can be none, any, or all of the following four groups, but only one from each group, and in the order specified below. (The exception to this rule is that ITEMized can be specified along with TABulated. If ITEMized is not specified, a TABulated vector list is treated by default as connected.)

1. BLOCK_normalized — All vectors will be normalized to a single common exponent.
2. Connectivity:
   - CONNECTED_lines — The first vector is an undisplayed position and the rest are endpoints of lines from the previous vector.
   - SEParate_lines — The vectors are paired as line endpoints.
   - DOTs — Each vector specifies a dot.
   - ITEMized — Each vector is individually specified as a move to position (P) or a line endpoint (L).
   - TABulated — This clause is used to specify an entry into a table that is used for specifying colors for raster lines and for specifying colors, radii, diffuse, and specular attributes for raster spheres. This option is also used to alter the attribute table itself.

When the TABulated option is used, the T=t clause replaces the I=i clause (for intensities) and the H=hue clause (for vector hues). The default is 127 (table entry 127).

ITEMized can be included with the TABulated option. If itemized is not specified, TABulated vector lists default to CONNECTED, where the first vector is an undisplayed position; all subsequent vectors represent endpoints of lines from the previous vector, regardless of P and L syntax.

There are 0 to 127 entries into the Attribute table. The Attribute table may be modified via input <14> of the SHADINGENVIRONMENT function.

3. Y and Z coordinate specifications (for constant or linearly changing Y and/or Z values):

```
Y = y[DY=delta_y][Z = z[DZ=delta_z]]
```

where y and z are default constants or beginning values, and delta_y and delta_z are increment values for subsequent vectors.

4. INTERNAL_units — Vector values are in the internal PS 390 units [LENGTH]. Specifying this option speeds the processing of the vector list, but this also requires P/L information to be specified for each vector, and it doesn't allow default Y values or specified intensities.

n — Estimated number of vectors.

vectors — The syntax for individual vectors will vary depending on the options specified in the options area. For all options except ITEMized and TABulated the syntax is:

```
xcomp[,ycomp[,zcomp]][I=i]
```

where xcomp, ycomp and zcomp are real or integer coordinates and i is a real number (0.0 < i < 1.0) specifying the intrinsic intensity for that point (1.0 = full intensity).

For ITEMized vector lists the syntax is:

```
P xcomp[,ycomp[,zcomp]][I=i]
```

or

```
L xcomp[,ycomp[,zcomp]][I=i]
```

where P means a move-to-position and L means a line endpoint.

If default y and z values are specified in the options area, they are not specified in the individual vectors.

For TABulated vector lists (TAB), the syntax is:

```
xcomp[,ycomp[,zcomp]][T=t]
```

where t is an integer between 0 and 127 specifying a table entry. Note that P and L can be specified if ITEMized is included with the TABulated option.

DEFAULTS

If not specified, the options default to:

1. block normalized
2. connected
3. no default Y or Z values are assumed (refer to Note 5)
4. Expecting internal units

Unspecifed vectors default to:

```
xcomp,ycomp[,zcomp][I=i]
```

If i is not specified, it defaults to 1.

Tabulated vectors default to:

```
xcomp,ycomp[,zcomp][T=t]
```

If the table entry is not specified, it defaults to 127 (table entry 127).

NOTES

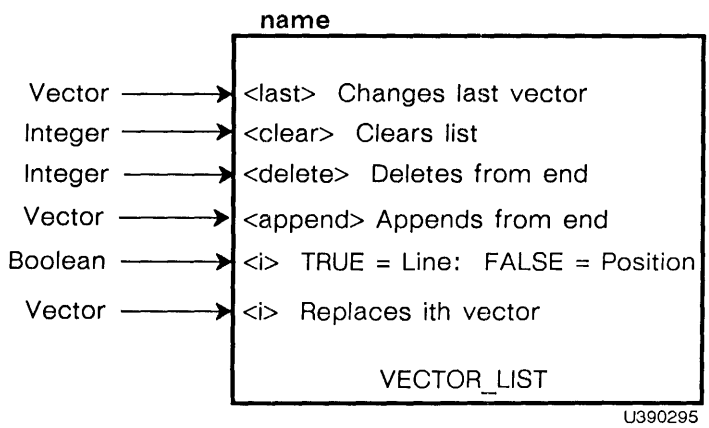
1. If n is less than the actual number of vectors, insufficient allocation of memory will result; if greater, more memory will be allocated than is used. (The former is generally the more severe problem.)

2. All vectors in a list must have the same number of components.
3. If y is specified in the options area, z must be specified in the options area.
4. If no default is specified in the options area and no Z components are specified in the vectors area, the vector list is a 2D vector list. If a Z default is specified in the same case, the vector list is a 3D vector list.
5. The first vector must be a position (P) vector and will be forced to be a position vector if not.
6. Options must be specified in the order given.
7. If CONNECTED_lines, SEParate_lines, or DOTS are specified in the options area but the vectors are entered using P/L, then the option specified takes precedence.
8. Block-normalized vector lists generally take longer to process into the PS 390, but are processed faster for display once they are in the system.

DISPLAY STRUCTURE NODE CREATED

Vector-list data node.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

1. Vector list nodes are in one of two forms:
   - If DOTS was specified in the options area of the command, a DOTS-mode vector-list node is created. The Boolean input to <i> is ignored in this case as well as the P/L portion of input vectors, and all vectors input are considered new positions for dots.
   - All other vector-list nodes created can be considered to be 2D or 3D ITEMized with intensity specifications after each vector, and if a 3D vector is input to a 2D vector-list node, the last component modifies the intensity.

2. If a 2D vector is sent to a 3D vector list, the Z value defaults to 0.

3. When you replace the ith vector, the new vector is considered a line (L) vector unless it was first changed to a position vector with F:POSITION_LINE.

EXAMPLES

```
A := VECtor_list BLOCK SEParate INTERNAL N=4
     P 1,1 L -1,1 L -1,-1 L 1,-1;

B := VECtor_list n=5
     1,1 -1,1 I=.5
     -1,-1 1,-1 I=.75
     1,1;

C := VECtor_list ITEM N=5
     P 1,1
     L -1,1
     L -1,-1
     P 1,-1
     L 1,1;

D := VECtor_list TABulated ITEM N=5  {for drawing raster lines}
     P 0,1,0
     L 0,0,0  t=5
     L 1,0,0  t=2
     P 1,1,0  t=3
     L 0,1,0  t=4;
```

TYPE

VIEWING — Viewport Specification

FORMAT

```
name := VIEWport HORizontal = hmin:hmax
                 VERTical = vmin:vmax
                 [INTENsity = imin:imax] [APPLied to name1];
```

DESCRIPTION

Specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines.

PARAMETERS

hmin,hmax,vmin,vmax — The X and Y boundaries of the new viewport. Values must be within the -1 to 1 range relative to the current viewport, implying that each viewport may be no larger than its predecessor.

imin,imax — Specifies the minimum and maximum intensities for the viewport. imin is the intensity of lines at the back clipping plane; imax at the front clipping plane. Values must be within the 0 to 1 range relative to the current viewport, implying that each viewport may have no greater intensity range than its predecessor.

name1 — Structure to which the viewport is applied.

DEFAULT

The initial viewport is the full PS 390 screen with full intensity range (0 to 1):

```
VIEWport HORizontal = -1:1 VERTical = -1:1 INTENsity = 0:1;
```

NOTES

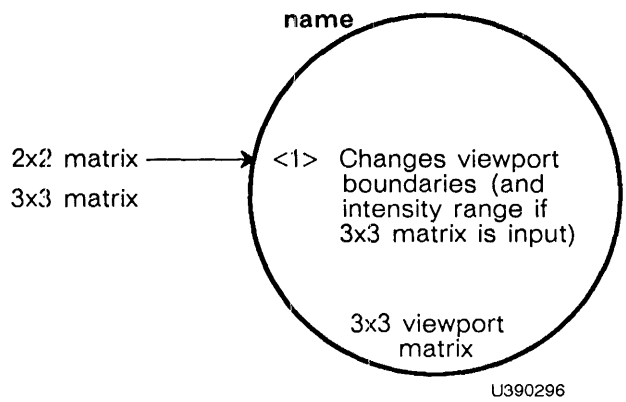
1. A new VIEWport is defined relative to the current viewport, whose boundaries are always taken to be -1:1 horizontally and vertically for the purposes of the command. (The "current" viewport is the one established by the most recent VIEWport command.)

2. Viewports can be nested to any level.
3. If the viewport aspect ratio (vertical/horizontal) is different from the window aspect ratio (Y/X) or field-of-view aspect ratio (always 1) being displayed in that viewport, the data displayed there will appear distorted.

DISPLAY STRUCTURE NODE CREATED

3x3 viewport-matrix operation node.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

1. For 2x2-matrix input, row 1 contains the hmin,hmax values and row 2 the vmin,vmax values.
2. For 3x3-matrix input, column 3 is ignored (there is no 3x2–matrix data type), rows 1 and 2 are as for the 2x2 matrix above, and row 3 contains the imin,imax values.

ASSOCIATED FUNCTIONS

F:MATRIX2, F:MATRIX3

EXAMPLE

```
A:= VIEWport HORizontal = 0:1
        VERTical = 0:1
        INTENsity = .5:1 THEN B;

B:= ... ;

{If A is displayed, structure B will be displayed in the upper right
quadrant of the screen with the intensity ranging from .5 to 1 instead
of 0 to 1.}
```

TYPE

VIEWING — Windowing Transformations

FORMAT

```
name := WINDOW X = xmin:xmax
              Y = ymin:ymax
              [FRONT boundary = zmin BACK boundary = zmax]
              [APPLied to name1];
```

DESCRIPTION

Specifies a right rectangular prism enclosing a portion of the world coordinate system to be displayed in parallel projection (compare Field_Of_View).

PARAMETERS

xmin...zmax — The window's boundaries along each axis (Refer to Note 3.)

name1 — Structure to which the window is applied.

DEFAULT

```
WINDOW X=-1:1 Y=-1:1 FRONT=0 BACK=100000;
```

NOTES

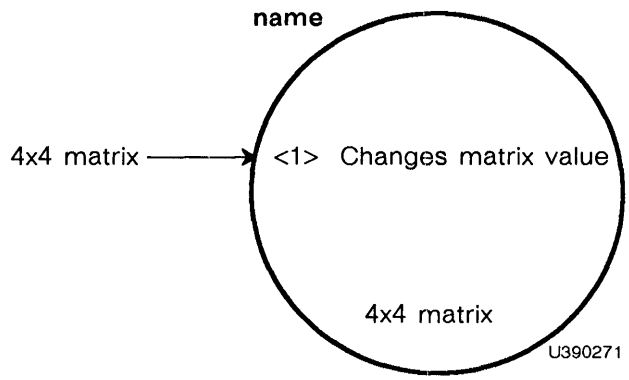
1. The windowing commands (WINDOW, Field_Of_View, and EYE) should always be the highest level element (the outermost transformation) in a display structure since these transformations override any previous transformations in the structure. Note that VIEWport is a mapping operation not a transformation of the data and thus is not affected by a windowing command.
2. These commands should also be followed by a LOOK command to fully specify the viewing transformation. (Refer to the LOOK command.)

3. The front and back boundaries should be specified relative to the AT point's position along the positive Z axis (0,0,D) (refer to the notes on the LOOK command). So, FRONT should equal (D minus delta_min) and BACK should equal (D plus delta_max), where delta_min and delta_max are the distances before and after the AT point that are to be included in the window, respectively. (Refer also to Note 3 of the LOOK command.)

DISPLAY STRUCTURE NODE CREATED

4x4-matrix operation node.

INPUT FOR UPDATING NODE



ASSOCIATED FUNCTIONS

F:WINDOW, F:FOV, F:MATRIX4

EXAMPLE

```
A:= BEGIN_Structure
    WINDOW X = -1:1 Y = -1:1
       FRONT boundary = 12
       BACK boundary = 14;
    LOOK AT 0,0,0 FROM 5,6.63,-10 THEN Sphere;
END_Structure;

{If Sphere is defined with a radius of 1 about the origin, A would be a
view of the Sphere from 5,6.63,-10, fully depth cued. Note that the
FROM to AT distance in the LOOK command is 13.}
```

TYPE

MODELING — Primitives

FORMAT

```
name := WITH PATtern i [AROUND_corners] [MATCH/NOMATCH] LENgth r
        VECtor_list;
```

DESCRIPTION

Uses line patterns (dashes, center lines, etc.) in drawing a vector list. The line pattern is created over the length r, so lines will have the pattern repeated as many times as necessary to the end of the line.

PARAMETERS

i — A series of up to 32 integers between 0 and 128 indicating the relative lengths of alternating lines, spaces, lines, etc., in the pattern. The longer the series, the more complex the pattern of lines and spaces, which repeats every r units.

AROUND_corners — indicates that patterning is to continue around each of the vectors in the vector list until the end of the list or a position vector is reached.

MATCH/NOMATCH — indicates that the pattern length should be adjusted to make the pattern exactly match the end points of the vector or series of vectors being patterned. The default is MATCH.

r — The length over which i is defined and repeated.

VECtor_list — indicates standard VECtor_list command with all options available except DOTs.

NOTES

1. The VECtor_list parameter n should be the estimate for the total number of vectors that will result from the command (not the number of vectors specified in the vector list).

2. As r approaches 0, n approaches infinity.
3. If r is greater than a vector line segment, that segment will be drawn solid; no pattern will be used.

DISPLAY STRUCTURE NODE CREATED

Vector list data node.

INPUTS FOR UPDATING NODE

See VECtor_list command.

NOTES ON INPUTS

Remember that the vectors in the node are the patterned vectors, so it is nontrivial to update a vector.

EXAMPLES

```
WITH PATTERN 1 1 LENgth 1 VECtor_list N=2 0,0 3,0;

WITH PATTERN 1 1 LENgth 3 VECtor_list N=2 0,0 3,0;

WITH PATTERN 1 1 LENgth 4 VECtor_list N=2 0,0 3,0;

WITH PATTERN 1 1 1 1 LENgth 2 VECtor_list N=2 0,0 3,0;

{same as the first example}

WITH PATTERN 1 .25 .125 .25 .125 .25 1 LENgth 3
VECtor_list N=2 0,0 3,0;
```

TYPE

MODELING — Transformed Data Attributes

FORMAT

```
name := WRITEBACK [APPLied to name1];
```

DESCRIPTION

The WRITEBACK command creates a WRITEBACK operation node and delineates the data structure below the node for writeback operations. When the WRITEBACK operation node is activated, writeback is performed for name1.

PARAMETER

name1 — The name of the structure or node to which writeback is applied.

NOTES

1. This node delimits the structure from which writeback data will be retrieved. Only the data nodes that are below the WRITEBACK operation node in the data structure will be transformed, clipped, viewport scaled, and sent back to the host.
2. Only a structure that is being displayed can be enabled for writeback. This means that the WRITEBACK operation node must be traversed by the display processor and so must be included in the displayed portion of the structure. If the writeback of only a portion of the picture is desired, WRITEBACK nodes must be placed appropriately in the display structure.
3. Any number of WRITEBACK nodes can be placed within a structure. Only one writeback operation can occur at a time. If more than one node is triggered, the writeback operations are performed in the order in which the corresponding nodes were triggered. If the user creates any WRITEBACK nodes (other than the WRITEBACK node created initially at boot-up), these nodes must be displayed before being triggered. If the nodes are triggered before being displayed, an error message will result.

4. Terminal emulator and Message_Display data will not be returned to the host.

DISPLAY STRUCTURE NODE CREATED

WRITEBACK operation node.

TYPE

MODELING — Transformed Data Attributes

FORMAT

```
name := XFORM output_data_type APPLied to name1;
```

DESCRIPTION

Allows transformed data to be saved either as a vector list or a 4x4 matrix at the point in the display structure where this XFORM data node is positioned.

PARAMETERS

output_data_type — Specifies what type of transformed data (MATRIX or VECTOR) is to be saved.

MATRIX — A single 4x4 matrix representing the concatenation of all transformation matrices currently in effect.

VECtor — A vector list specifying the transformed coordinates of the object (name1).

name1 — The object whose transformed data are to be saved.

NOTE

This node indicates to the F:XFORMDATA function the point in the display structure where transformed data are requested.

DISPLAY STRUCTURE NODE CREATED

XFORM operation node.

ASSOCIATED FUNCTIONS

F:XFORMDATA, F:LIST, F:SYNC(2).

EXAMPLE

```
Xform := BEGIN_Structure  {Set up switch mechanism}
         X := SET Conditional_BIT 1 ON;
         IF conditional_BIT 1 is ON THEN VIEW;
         IF conditional_BIT 1 is OFF THEN TRAN;
       END_Structure;

Tran := BEGIN_Structure  {To be used while getting transformed data}
         Matrix_4x4 1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1;
         INSTance of Obj;
        END_Structure;

View := BEGIN_Structure  {To be used while viewing and designing}
                            {Viewing commands: Field_Of_View, WINDOW
                             EYE BACK, or 4x4_Matrix}
         INSTance of Obj;
         END_Structure;

Obj := BEGIN_Structure  {Setup transformed-data request}
       {Transformation commands:
       ROTATE, TRANSLATE, and/or SCALE}
       Xform_Request := XFORM VECtor;
       INSTance of Data;
       END_Structure;

XformData := F:XFORMDATA;  {Build transformed-data network}
Sync2 := F:SYNC(2);
List := F:LIST;
CONN Sync2<1>:<1>XFORMDATA;
CONN XformData<1>:<1>List;
CONN List<1>:<1>HOST_MESSAGE;  {Send transformed data to host}
CONN List<2>:<2>Sync2;  {"Task completed" flag}
SEND <any message> TO <2>Sync2;
SEND 'OBJ.XFORM_REQUEST' TO <2>XformData;
SEND 'XDATA' TO <3>XformData;
DISPLAY Xform;
```

Appendix A
PS 390 Commands by Category

Advanced Programming (Memory Allocation)

RAWBLOCK

Function (Data Structuring)

VARIABLE

Function (Immediate Action)

CONNECT

DISCONNECT

SEND

SEND number*mode

SEND VL

SETUP CNESS

STORE

General (Immediate Action - Command Control and Status)

BEGIN...END

COMMAND STATUS

CONFIGURE

FINISH CONFIGURATION

GIVE_UP_CPU

OPTIMIZE MEMORY

OPTIMIZE STRUCTURE...END OPTIMIZE

REBOOT

RESERVE_WORKING_STORAGE

!RESET

SET PRIORITY

SETUP INTERFACE

SETUP PASSWORD

SHOW INTERFACE

General (Immediate Action - Data Structuring and Display)

DELETE

DISPLAY

FORGET (structures)

FORGET (units)

REMOVE

General (Immediate Action - Initialization)

INITIALIZE

Modeling (Data Structuring - Character Font)

BEGIN_FONT...END_FONT

CHARACTER FONT

STANDARD FONT

Modeling (Data Structuring - Character Transformations)

CHARACTER ROTATE

CHARACTER SCALE

MATRIX_2x2

TEXT SIZE

Modeling (Data Structuring - Line Pattern)

PATTERN

PATTERN WITH

SET LINE_TEXTURE

Modeling (Data Structuring - Picking Attributes)

SET PICKING

SET PICKING IDENTIFIER

SET PICKING LOCATION

Modeling (Data Structuring - Primitives)

BSPLINE

CHARACTERS

COPY

ERASE PATTERN FROM

LABELS

PATTERN

PATTERN WITH

POLYGON

POLYNOMIAL

RATIONAL BSPLINE

RATIONAL POLYNOMIAL

VECTOR_LIST

WITH PATTERN

Modeling (Data Structuring - Transformed Data Attributes)

CANCEL XFORM

WRITEBACK

XFORM

Modeling (Data Structuring - Transformations)

MATRIX_3x3

MATRIX_4x3

MATRIX_4x4

ROTATE

SCALE

TRANSLATE

Rendering (Data Structuring)

ATTRIBUTES

ILLUMINATION

SECTIONING_PLANE

SOLID_RENDERING

SURFACE_RENDERING

Structure (Data Structuring - Attributes)

DECREMENT LEVEL_OF_DETAIL

INCREMENT LEVEL_OF_DETAIL

SET BLINKING ON/OFF

SET BLINK RATE

SET CONDITIONAL_BIT

SET LEVEL_OF_DETAIL

SET RATE

SET RATE EXTERNAL

Structure (Data Structuring - Conditional Referencing)

IF CONDITIONAL_BIT

IF LEVEL_OF_DETAIL

IF PHASE

Structure (Data Structuring - Explicit Referencing)

APPLIED TO/THEN

(Function Instancing)

INSTANCE OF

(Naming of Display Structure Nodes)

NIL

Structure (Data Structuring - Implicit Referencing)

BEGIN_STRUCTURE...END_STRUCTURE

Structure (Immediate-action - Modifying)

FOLLOW WITH

INCLUDE

PREFIX WITH

REMOVE FOLLOWER

REMOVE FROM

REMOVE PREFIX

Viewing (Data Structuring - Appearance Attributes)

SELECT FILTER

SET CHARACTERS

SET COLOR

SET CONTRAST

SET DEPTH_CLIPPING

SET DISPLAYS

Viewing (Data Structuring - Viewport Specification)

LOAD VIEWPORT

SET INTENSITY

VIEWPORT

Viewing (Data Structuring - Windowing Transformations)

EYE

FIELD_OF_VIEW

LOOK

WINDOW

Appendix B
PS 390 Command Syntax

APPLIED TO/THEN

```
name := operation_command [APPLied to name1];
name := operation_command [THEN name1];
```

ATTRIBUTES

```
name := ATTRIBUTES attributes [AND attributes];
```

BEGIN...END

```
BEGIN
command;
command;
.
.
.
command;
END;
```

BEGIN_FONT...END_FONT

```
name := BEGIN_Font
        [C[0]: N=n {itemized 2D vectors};]
                .
                .
                .
        [C[i]: N=n {itemized 2D vectors};]
                .
                .
                .
        [C[127]: N=n {itemized 2D vectors};]
      END_Font;
```

BEGIN_S...END_S

```
name :=  BEGIN_Structure
[name1:=]    nameable_command;
                  .
                  .
                  .
 [namen:=]    nameable_command;
      END_Structure;
```

BSPLINE

```
name := BSpline ORDER=k
        [OPEN/CLOSED] [NONPERIodic/PERIodic] [N=n]
        [VERTICES =] x1,y1,[z1]
                x2,y2,[z2]
                       .  .  .
                       .  .  .
                       .  .  .
                xn,yn,[zn]
        [KNOTS = t1,t2,...,tj]
        CHORDS = q;
```

CANCEL XFORM

```
name := CANCEL XFORM [APPLied to name1];
```

CHARACTER FONT

```
name := character FONT font_name [APPLied to name1];
```

CHARACTER ROTATE

```
name := CHARacter ROTate angle [APPLied to name1];
```

CHARACTERS

```
name := CHARacters [x,y[,z]][STEP dx,dy] 'string';
```

CHARACTER SCALE

```
name := CHARacter SCAle s [APPLied to name1];
name := CHARacter SCAle sx,sy [APPLied to name1];
```

COMMAND STATUS

```
COMmand STATus;
```

CONFIGURE

```
CONFIGURE password;
```

CONNECT

```
CONNect name1<i>:<j>name2;
```

COPY

```
name := COPY name1 [START=] i [,] [COUNT=] n;
```

DECREMENT LEVEL_OF_DETAIL

```
name:= DECrement LEVel_of_detail[APPLied to name1];
```

DELETE

```
DELete name[,name1 ... namen];
DELete any_string*;
```

DISCONNECT

```
DISCONNect name1[<i>]:option;
DISCONNect name1<i>:<j>name2;
```

DISPLAY

```
DISPlay name;
```

ERASE PATTERN FROM

```
ERASE PATTERN FROM name;
```

EYE BACK

```
name := EYE BACK z [option1][option2] from SCREEN area w WIDE
          [FRONT boundary = zmin BACK boundary = zmax]
          [APPLied to name1];
```

FIELD_OF_VIEW

```
name := Field_Of_View angle
          [FRONT boundary = zmin BACK boundary = zmax]
          [APPLied to name1];
```

FINISH CONFIGURATION

```
FINISH CONFIGURATION;
```

FOLLOW WITH

```
FOLLOW name WITH option;
```

FORGET (structures)

```
FORget name;
```

FORGET (units)

```
FORget (unit_name);
```

(Function Instancing)

```
name := F:function_name;
```

GIVE_UP_CPU

```
GIVE_UP_CPU;
```

IF CONDITIONAL_BIT

```
name := IF conditional_BIT n is state [THEN name1];
```

IF LEVEL_OF_DETAIL

```
name := IF LEVel_of_detail relationship n [THEN name1];
```

IF PHASE

```
name := IF PHASE is state THEN [name1];
```

ILLUMINATION

```
name := ILLUMINATION x,y,z [COLOR h [,s [,i]]] [AMBIENT a];
```

INCLUDE

```
INCLude name1 IN name2;
```

INCREMENT LEVEL_OF_DETAIL

```
name:= INCRement LEVel_of_detail[APPLied to name1];
```

INITIALIZE

```
INITialize [option];
```

INSTANCE OF

```
name := INSTance of name1[,name2 ... namen];
```

LABELS

```
name := LABELS x,y,[,z] 'string'
                        .
                        .
                [xi,yi [,zi] 'string'];
```

LOAD VIEWPORT

```
name := LOAD VIEWport HORizontal = hmin:hmax
              VERTical = vmin:vmax
              [INTENsity = imin:imax] [APPLied to name1];
```

LOOK

```
name := LOOK AT ax,ay,az FROM fx,fy,fz
          [UP ux,uy,uz] [APPLied to name1];

name := LOOK FROM fx,fy,fz AT ax,ay,az
          [UP ux,uy,uz] [APPLied to name1];
```

MATRIX_2x2

```
name := Matrix_2x2 m11,m12
                   m21,m22 [APPLied to name1];
```

MATRIX_3x3

```
name := Matrix_3x3 m11,m12,m13
                   m21,m22,m23
                   m31,m32,m33 [APPLied to name1];
```

MATRIX_4x3

```
name := Matrix_4x3 m11,m12,m13
                   m21,m22,m23
                   m31,m32,m33
                   m41,m42,m43 [APPLied to name1];
```

MATRIX_4x4

```
name := Matrix_4x4 m11,m12,m13,m14
                   m21,m22,m23,m24
                   m31,m32,m33,m34
                   m41,m42,m43,m44 [APPLied to name1];
```

(Naming of Display Structure Nodes)

```
name:= display_structure_command;
```

NIL

```
name := NIL;
```

OPTIMIZE MEMORY

```
OPTIMIZE MEMORY;
```

OPTIMIZE STRUCTURE; .... END OPTIMIZE;

```
OPTIMIZE STRUCTURE;
         command;
         command;
            .
            .
 END OPTIMIZE;
```

PATTERN

```
name := PATtern i [AROUND_corners][MATCH/NOMATCH] LENgth r;
```

PATTERN WITH

```
PATTERN name1 WITH pattern;
```

POLYGON

```
name := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [COPLANAR]
        POLYGon vertex ... vertex;
```

POLYNOMIAL

```
name:= POLYnomial[ORDER=i]
        [COEFFICIENTS=]  xi,  yi,   zi
                         xi-1, yi-1, zi-1
                           .     .     .
                           .     .     .
                           .     .     .
                          x0,   y0,   z0
        CHORDS= q;
```

PREFIX WITH

```
PREFIX name WITH operation_command;
```

RATIONAL BSPLINE

```
name := RATIonal BSpline ORDER=k
        [OPEN/CLOSED] [NONPERIodic/PERIodic] [N=n]
        [VERTICES =]   x1,y1,[z1],w
                       x2,y2,[z2],w2
                        .    .    .    .
                        .    .    .    .
                        .    .    .    .
                       xn,yn,[zn],wn
         [KNOTS = t1,t2,...,tj]
         CHORDS = q;
```

RATIONAL POLYNOMIAL

```
name:= RATional POLYnomial[ORDER=i]
       [COEFFICIENTS=] xi,   yi,   zi,   wi
                       xi-1, yi-1, zi-1, wi-1
                          .     .     .     .
                          .     .     .     .
                          .     .     .     .
                       x0,   y0,   z0,   w0
        CHORDS= q;
```

RAWBLOCK

```
name := RAWBLOCK i;
```

REBOOT

```
name := REBOOT password;
```

REMOVE

```
REMove name;
```

REMOVE FOLLOWER

```
REMove FOLLOWER of name;
```

REMOVE FROM

```
REMove name1 FROM name2;
```

REMOVE PREFIX

```
REMove PREfix of name;
```

RESERVE_WORKING_STORAGE

```
RESERVE_WORKING_STORAGE size;
```

!RESET

```
!RESET;
```

ROTATE

```
name := ROTate in [axis] angle [APPLied to name1];
```

SCALE

```
name := SCALE by s [APPLied to name1];
name := SCALE by sx,sy[,sz] [APPLied to name1];
```

SECTIONING_PLANE

```
name := SECTioning_plane [APPLied to name1];
```

SELECT FILTER

```
name1 := SELECT FILTER n THEN Name2;
```

SEND

```
SEND option TO <n>name1;
```

SEND number*mode

```
SEND number*mode TO <n>name1;
```

SEND VL

```
SEND VL(name1) TO <i>name2;
```

SET BLINKING ON/OFF (PS 350)

```
name := SET BLINKing switch [APPLied to name1];
```

SET BLINK RATE

```
name := SET BLINK RATE n [APPLied to name1];
```

SET CHARACTERS

```
name := SET CHARacters orientation [APPLied to name1];
```

SET COLOR

```
name := SET COLOR hue,sat [APPLied to name1];
```

SET CONDITIONAL_BIT

```
name := SET conditional_BIT n switch [APPLied to name1];
```

SET CONTRAST

```
name := SET CONTrast to c [APPLied to name1];
```

SET DEPTH_CLIPPING

```
name := SET DEPTH_CLipping switch [APPLied to name1];
```

SET DISPLAYS

```
name := SET DISPlays ALL switch [APPLied to name1];
name := SET DISPlay n[,m...] switch [APPLied to name1];
```

SET INTENSITY

```
name := SET INTENsity switch imin:imax [APPLied to name1];
```

SET LEVEL_OF_DETAIL

```
name := SET LEVel_of_detail to n [APPLied to name1];
```

SET LINE_TEXTURE

```
name := SET LINe_texture [AROUnd_corners] pattern
        [APPLied to name1];
```

SET PICKING

```
name := SET PICKing switch [APPLied to name1];
```

SET PICKING IDENTIFIER

```
name := SET PICKing IDentifier = id_name
        [APPLied to name1];
```

SET PICKING LOCATION

```
name := SET PICKing LOCation = x,y size_x,size_y;
```

SET PRIORITY

```
Set Priority of name to i;
```

SET RATE

```
name := SET RATE phase_on phase_off [initial_state] [delay]
        [APPLied to name1];
```

SET RATE EXTERNAL

```
name:= SET RATE EXTernal [APPLied to name1];
```

SETUP CNESS

```
SETUP CNESS queue_type <i>name;
```

SETUP INTERFACE

```
SETUP INTERFACE portn/option=<n>
```

SETUP PASSWORD

```
SETUP PASSWORD password;
```

SHOW INTERFACE

```
SHOW INTERFACE <name>;
```

SOLID_RENDERING

```
name := SOLID_rendering [APPLied to name1];
```

STANDARD FONT

```
name := STANdard FONT [APPLied to name1];
```

STORE

```
STORE option IN name1;
```

SURFACE_RENDERING

```
name := SURFACE_rendering [APPLied to name1];
```

TEXT SIZE

```
name := TEXT SIZE x [APPLIED to name1];
```

TRANSLATE

```
name := TRANslate by tx,ty[,tz] [APPLied to name1];
```

VARIABLE

```
VARiable name1[,name2 ... namen];
```

VECTOR_LIST

```
name := VECtor_list [options] [N=n] vectors;
```

VIEWPORT

```
name := VIEWport HORizontal = hmin:hmax
        VERTical = vmin:vmax
        [INTENsity = imin:imax] [APPLied to name1];
```

WINDOW

```
name := WINDOW X = xmin:xmax Y = ymin:ymax
        [FRONT boundary = zmin BACK boundary = zmax]
        [APPLied to name1];
```

WITH PATTERN

```
name := WITH PATtern i [AROUND_corners][MATCH/NOMATCH]
        LENgth r VECtor_list;
```

WRITEBACK

```
name := WRITEBACK [APPLied to name1];
```

XFORM

```
name := XFORM output_data_type [APPLied to name1];
```

Appendix C
ASCII Commands and Corresponding GSRs

This appendix contains a list of the PS 390 ASCII commands and the corresponding FORTRAN, Pascal and UNIC/C GSRs. The names of the utility and raster routines and the corresponding GSRs are also included. GSR descriptions will be found in Section RM4.

The user should note the following when using this appendix:

The left column lists the ASCII command or routine name in alphabetical order. The right three columns list the corresponding FORTRAN, Pascal and UNIX/C GSR. N/A means that there is no GSR.

In general, there is a one-to-one correspondence between ASCII commands and the corresponding GSRs. The following three ASCII commands require more than one GSR:

- LABELS
- POLYGON
- VECTOR_LIST

The utility and raster routines do not have a corresponding ASCII command.

ASCII commands with different parameters have separate GSRs. The name of the corresponding ASCII command will contain the parameter. For example, the ROTATE command has the following three corresponding GSRs:

- ROTATE IN X
- ROTATE IN Y
- ROTATE IN Z

ASCII Command/Routine Name	FORTRAN	Pascal	UNIX/C
Attach PS 390 to Communication Device - utility GSR	PAttch	PAttach	PAttach
ATTRIBUTES	PAttr	PAttrib	PAttrib
ATTRIBUTES	PAttr2	PAttrib2	PAttrib2
BEGIN...END	PBeg	PBegin	PBegin
BEGIN...END	PEnd	PEnd	PEnd
Begin Saving GSR Data - utility GSR	N/A	N/A	PSavBeg
BEGIN_STRUCTURE...END_STRUCTURE	PBegS	PBeginS	PBeginS
BEGIN_STRUCTURE...END_STRUCTURE	PEndS	PEndS	PEndS
BSPLINE	PBspl	PBspl	PBspl
CANCEL XFORM	PXfCan	PXfCancl	PXfCancl
CHARACTER FONT	PFont	PFont	PFont
CHARACTER ROTATE	PChRot	PCharRot	PCharRot
CHARACTERS	PChs	PChars	PChars
CHARACTER SCALE	PChSca	PCharSca	PCharSca
CONNECT	PConn	PConnect	PConnect
Convert HSI to RGB - utility GSR	PSURGB	PSUTIL_HSIRGB	PSUTIL_HSIRGB
COPY	PCopyV	PCopyVec	PCopyVec
DECREMENT LEVEL_OF_DETAIL	PDeLOD	PDecLOD	PDecLOD
DELETE	PDelet	PDelete	PDelete
DELETE STRING	PDelW	PDelWild	PDelWild
Detach PS 390 from Communication Device - utility GSR	PDtach	PDetach	PDetach
DISCONNECT	PDi	PDisc	PDisc
DISCONNECT ALL	PDiAll	PDiscAll	PDiscAll
DISCONNECT OUTPUT	PDiOut	PDiscOut	PDiscOut

(continued)

ASCII Command/Routine Name	FORTRAN	Pascal	UNIX/C
DISPLAY	PDisp	PDisplay	PDisplay
End Saving GSR Data - utility GSR	N/A	N/A	PSavEnd
ERASE PATTERN FROM	PEraPa	PEraPatt	PEraPatt
Erase Screen - raster GSR	PRasEr	PRasEr	PRasEr
EYE BACK	PEyeBk	PEyeBack	PEyeBack
FIELD_OF_VIEW	PFov	PFov	PFov
FOLLOW WITH	PFoll	PFoll	PFoll
FORGET (Structures)	PForg	PForget	PForget
(Function Instancing)	PFn	PFnInst	PFnInst
(Function Instancing)	PFnN	PFnInstN	PFnInstN
GIVE_UP_CPU	PGUCPU	PGiveUpCPU	PGiveUpCPU
IF CONDITIONAL_BIT	PIfBit	PIfBit	PIfBit
IF LEVEL_OF_DETAIL	PIfLev	PIfLevel	PIfLevel
IF PHASE	PIfPha	PIfPhase	PIfPhase
ILLUMINATION	PIllum	PIllumin	PIllumin
INCLUDE	PIncl	PIncl	PIncl
INCREMENT LEVEL_OF_DETAIL	PInLOD	PIncLOD	PIncLOD
INITIALIZE	PInit	PInit	PInit
INITIALIZE CONNECTIONS	PInitC	PInitC	PInitC
INITIALIZE DISPLAY	PInitD	PInitD	PInitD
INITIALIZE NAMES	PInitN	PInitN	PInitN
INSTANCE OF	PInst	PInst	PInst

(continued)

ASCII Command/Routine Name	FORTRAN	Pascal	UNIX/C
LABELS	PLaAdd	PLabAdd	PLabAdd
	PLaBeg	PLabBegn	PLabBegn
	PLaEnd	PLabEnd	PLabEnd
Load Pixel Data - raster GSR	PRasWP	PRasWP	PRasWP
Load Saved GSR Data - utility GSR	N/A	N/A	PLoad
LOOK	PLookA	PLookAt	PLookAt
MATRIX_2X2	PMat22	PMat2x2	PMat2x2
MATRIX_3X3	PMat33	PMat3x3	PMat3x3
MATRIX_4X3	PMat43	PMat4x3	PMat4x3
MATRIX_4X4	PMat44	PMat4x4	PMat4x4
NIL	PNil	PNameNil	PNameNil
OPTIMIZE STRUCTURE;...END OPTIMIZE;	POpt	POptStru	POptStru
OPTIMIZE STRUCTURE;...END OPTIMIZE;	PEndOp	PEndOpt	PEndOpt
PATTERN	PDefPa	PDefPatt	PDefPatt
PATTERN WITH	PPatWi	PPatWith	PPatWith
Poll PS 390 for Messages - utility GSR	PGet	PGet	PGet
POLYGON	PPlygA	PPlygAtr	PPlygAtr
	PPlygB	PPlygBeg	PPlygBeg
	PPlygE	PPlygEnd	PPlygEnd
	PPlygH	PPlygHSI	PPlygLisHSI
	PPlygL	PPlygLis	PPlygLis
	PPlygO	PPlygOtl	PPlygOtl
	PPlygR	PPlygRGB	PPlygLisRGB
POLYNOMIAL	PPoly	PPoly	PPoly
PREFIX WITH	PPref	PPref	PPref
Purge Output Buffer - utility GSR	PPurge	PPurge	PPurge

(continued)

ASCII Command/Routine Name	FORTRAN	Pascal	UNIX/C
Query GSR Device Status - utility GSR	PDInfo	PDevInfo	N/A
RATIONAL BSPLINE	PRBspl	PRBspl	PRBspl
RATIONAL POLYNOMIAL	PRPoly	PRPoly	PRPoly
RAWBLOCK	PRawBl	PRawBloc	PRawBloc
Read Messages from PS 390 - utility GSR	PGetW	PGetWait	PGetWait
REMOVE	PRem	PRem	PRem
REMOVE FOLLOWER	PRemFo	PRemFoll	PRemFoll
REMOVE FROM	PRemFr	PRemFrom	PRemFrom
REMOVE PREFIX	PRemPr	PRemPref	PRemPref
RESERVE_WORKING_STORAGE	PRsvSt	PRsvStor	PRsvStor
ROTATE IN X	PRotX	PRotX	PRotX
ROTATE IN Y	PRotY	PRotY	PRotY
ROTATE IN Z	PRotZ	PRotZ	PRotZ
SCALE	PScale	PScaleBy	PScaleBy
SECTIONING_PLANE	PSecPl	PSecPlan	PSecPlan
SEND 2D MATRIX	PSnM2d	PSndM2d	PSndM2d
SEND 2D VECTOR	PSnV2d	PSndV2d	PSndV2d
SEND 3D MATRIX	PSnM3d	PSndM3d	PSndM3d
SEND 3D VECTOR	PSnV3d	PSndV3d	PSndV3d
SEND 4D MATRIX	PSnM4d	PSndM4d	PSndM4d
SEND 4D VECTOR	PSnV4d	PSndV4d	PSndV4d

(continued)

ASCII Command/Routine Name	FORTRAN	Pascal	UNIX/C
SEND BOOLEAN	PSnBoo	PSndBool	PSndBool
Send Bytes to Generic Output Channel - utility GSR	PPutG	PPutG	PPutG
Send Bytes to Generic Output Channel - utility GSR	PPutGX	PPutGX	N/A
Send Bytes to Parser Output Channel - utility GSR	PPutP	PPutPars	PPutPars
SEND FIX	PSnFix	PSndFix	PSndFix
SEND number*mode	PSnPL	PSndPL	PSndPL
SEND RAW STRING	PSnRSt	PSndRStr	N/A
SEND REAL NUMBER	PSnRea	PSndReal	PSndReal
SEND STRING	PSnSt	PSndStr	PSndStr
SEND VALUE	PSnVal	PSndVal	PSndVal
SEND VL	PSnVL	PSndVL	PSndVL
SET CHARACTERS SCREEN_ORIENTED	PSeChS	PSetChrS	PSetChrS
SET CHARACTERS SCREEN_ORIENTED/FIXED	PSeChF	PSetChrF	PSetChrF
SET CHARACTERS WORLD_ORIENTED	PSeChW	PSetChrW	PSetChrW
SET COLOR	PSeCol	PSetColr	PSetColr
SET CONDITIONAL_BIT	PSeBit	PSetBit	PSetBit
SET CONTRAST	PSeCon	PSetCont	PSetCont
Set Current Pixel Location - raster GSR	PRasCp	PRasCp	PRasCp
Set Delimiting Character - utility GSR	PDelim	N/A	N/A
SET DEPTH_CLIPPING	PSeDCL	PSetDCL	PSetDCL

(continued)

ASCII Command/Routine Name	FORTRAN	Pascal	UNIX/C
SET DISPLAY	PSeDOF	PSetDOnF	PSetDOnF
SET DISPLAYS ALL	PSeDAl	PSetDAll	PSetDAll
Set Global Binary Output Channel - utility GSR	PMuxCI	PMuxCI	PMuxCI
Set Global Generic Channel - utility GSR	PMuxG	PMuxG	PMuxG
Set Global Parser Channel - utility GSR	PMuxP	PMuxPars	PMuxPars
SET INTENSITY	PSeInt	PSetInt	PSetInt
SET LEVEL_OF_DETAIL	PSeLOD	PSetLOD	PSetLOD
SET LINE_TEXTURE	PSeLnT	PSetLinT	PSetLinT
Set Logical Device Coordinates - raster GSR	PRasLd	PRasLd	PRasLd
SET PICKING	PSePOf	PSetPOnf	PSetPOnf
SET PICKING IDENTIFIER	PSePID	PSetPID	PSetPID
SET PICKING LOCATION	PSePLo	PSetPLoc	PSetPLoc
Set Raster Mode to Write Pixel Data - raster GSR	PRaWRP	PRaWRP	PRaWRP
SET RATE	PSeR	PSetR	PSetR
SET RATE EXTERNAL	PSeREx	PSetRExt	PSetRExt
SETUP CNESS	PseCns	PSetCnes	PSetCnes
SOLID_RENDERING	PSolRe	PSolRend	PSolRend
STANDARD FONT	PStdFo	PStdFont	PStdFont
SURFACE_RENDERING	PSurRe	PSurRend	PSurRend
TRANSLATE	PTrans	PTransBy	PTransBy

(continued)

ASCII Command/Routine Name	FORTRAN	Pascal	UNIX/C
VARIABLE	PVar	PVar	PVar
VECTOR_LIST	PVcBeg	PVecBegn	PVecBegn
	PVcEnd	PVecEnd	PVecEnd
	PVcLis	PVecLis	PVecLis
	PVcMax	PVecMax	PVecMax
VIEWPORT	PViewP	PViewP	PViewP
WINDOW	PWindo	PWindow	PWindow
WRITEBACK	PWrtBk	PWrtBack	PWrtBack
XFORM MATRIX	PXfMat	PXfMatrx	PXfMatrx
XFORM VECTOR_LIST	PXfVec	PXfVectr	PXfVectr

ASCII Character Code Set

Decimal Value	ASCII Character	Decimal Value	ASCII Character	Decimal Value	ASCII Character
0	NUL	44	'	88	X
1	SOH	45	-	89	Y
2	STX	46	.	90	Z
3	ETX	47	/	91	[
4	EOT	48	0	92	\
5	ENQ	49	1	93	]
6	ACK	50	2	94	↑ or ^
7	BEL	51	3	95	← or _
8	BS	52	4	96	`
9	HT	53	5	97	a
10	LF	54	6	98	b
11	VT	55	7	99	c
12	FF	56	8	100	d
13	CR	57	9	101	e
14	SO	58	:	102	f
15	SI	59	;	103	g
16	DLE	60	<	104	h
17	DC1	61	=	105	i
18	DC2	62	>	106	j
19	DC3	63	?	107	k
20	DC4	64	@	108	l
21	NAK	65	A	109	m
22	SYN	66	B	110	n
23	ETB	67	C	111	o
24	CAN	68	D	112	p
25	EM	69	E	113	q
26	SUB	70	F	114	r
27	ESC or ALT	71	G	115	s
28	FS	72	H	116	t
29	GS	73	I	117	u
30	RS	74	L	118	v
31	VS	75	K	119	w
32	SP	76	L	120	x
33	!	77	M	121	v
34	"	78	N	122	z
35	#	79	O	123	{
36	$	80	P	124	|
37	%	81	Q	125	}
38	&	82	R	126	~ TILDE
39	´	83	S	127	Rubout or DEL
40	(	84	T		
41	)	85	U		
42	*	86	V		
43	+	87	W		

RM2. INTRINSIC FUNCTIONS

RM2. INTRINSIC FUNCTIONS

CONTENTS

1. INTRINSIC FUNCTIONS 1

2. FUNCTION REPRESENTATION 2

3. CONJUNCTIVE/DISJUNCTIVE SETS 3

4. CONSTANT AND ACTIVE QUEUES 4

5. QUEUE DATA TYPES 6

6. INTRINSIC USER FUNCTIONS 7

F:ACCUMULATE 8
F:ADD 11
F:ADDC 12
F:ALLOW_VECNORM 13
F:AND 14
F:ANDC 15
F:ATSCALE 16
F:AVERAGE 18
F:BOOLEAN_CHOOSE 19
F:BROUTE 20
F:BROUTEC 21
F:CBROUTE 22
F:CCONCATENATE 23
F:CDIV 24
F:CEILING 25
F:CGE 26
F:CGT 27
F:CHANGEQTYPE 28

F:CHARCONVERT 29
F:CHARMASK 31
F:CHOP 32
F:CI(n) 33
F:CIROUTE(n) 35
F:CLCSECONDS 37
F:CLE 39
F:CLFRAMES 40
F:CLT 42
F:CLTICKS 43
F:CMUL 45
F:COMP_STRING 46
F:CONCATENATE 47
F:CONCATENATEC 48
F:CONCATXDATA(n) 49
F:CONSTANT 50
F:CROTATE 51
F:CROUTE(n) 52
F:CSCALE 53
F:CSUB 54
F:CVEC 55
F:CVT6TO8 56
F:CVT8TO6 57
F:CVTASCTOIBM 58
F:CVTIBMTOASC 59
F:DELTA 60
F:DEMUX(n) 61
F:DEPACKET 63
F:DIV 65
F:DIVC 66
F:DSCALE 67
F:DXROTATE 69
F:DYROTATE 70
F:DZROTATE 71
F:EDGE_DETECT 72
F:EQ 73
F:EQC 74
F:FCNSTRIP 75
F:FETCH 76
F:FIND_STRING 77
F:FIX 78
F:FLOAT 79

F:FOV 80
F:GATHER_GENFCN 82
F:GATHER_STRING 83
F:GE 84
F:GEC 85
F:GT 86
F:GTC 87
F:HOLDMESSAGE 88
F:INPUTS_CHOOSE(n) 90
F:LABEL 91
F:LBL_EXTRACT 92
F:LE 93
F:LEC 94
F:LENGTH_STRING 95
F:LIMIT 96
F:LINEEDITOR 98
F:LIST 101
F:LOOKAT 102
F:LOOKFROM 103
F:LT 104
F:LTC 105
F:MAKEPACKET 106
F:MATRIX2 107
F:MATRIX3 108
F:MATRIX4 109
F:MCAT_STRING(n) 110
F:MINMAX(n) 111
F:MOD 112
F:MODC 113
F:MUL 114
F:MULC 115
F:MUX 116
F:NE 117
F:NEC 118
F:NOP 119
F:NOT 120
F:NPRT_PRT 121
F:OR 122
F:ORC 123
F:PACKET 124
F:PARTS 126
F:PASSTHRU(n) 127

F:PICKINFO 128
F:POSITION_LINE 131
F:PRINT 132
F:PUT_STRING 136
F:RANGE_SELECT 137
F:READDISK 139
F:READSTREAM 140
F:RESET 141
F:ROUND 142
F:ROUTE(n) 143
F:ROUTEC(n) 144
F:SCALE 145
F:SCREENSAVE 146
F:SEND 147
F:SINCOS 148
F:SPLIT 149
F:SQROOT 150
F:STRING_TO_NUM 151
F:SUB 152
F:SUBC 153
F:SYNC(n) 154
F:TAKE_STRING 156
F:TIMEOUT 157
F:TRANS_STRING 159
F:VEC 160
F:VECC 161
F:VEC_EXTRACT 162
F:WINDOW 163
F:WRITEDISK 165
F:WRITESTREAM 166
F:XFORMDATA 167
F:XOR 170
F:XORC 171
F:XROTATE 172
F:XVECTOR 173
F:YROTATE 174
F:YVECTOR 175
F:ZROTATE 176
F:ZVECTOR 177

7. INTRINSIC SYSTEM FUNCTIONS 178

F:F_I1_IBM
F:F_I2_IBM 179

F:F_W_IBM 180
F:IBMDISP 181
F:IBM_KEYBOARD 182
F:K2ANSI 184
F:RASTER 185
F:RASTERSTREAM 186
F:SETUPIBM 187
F:STATDIS 188
F:TEDUP 189
F:USRTOF 190
F:VT10 191

APPENDIX A
INTRINSIC FUNCTIONS BY CATEGORY 192

Classification of Functions 192
Intrinsic User Functions 193
Intrinsic System Functions 196
ASCII Character Code Set 197

Section RM2
Intrinsic Functions

A function is the processing component of a function network. It performs one or more operations by accepting input, processing that input, and producing output. In the PS 390 there are two types of functions: intrinsic functions and user-written functions.

There are two types of intrinsic functions: intrinsic user functions and intrinsic system functions. Intrinsic user functions may be instanced by a user to create a function network. Intrinsic system functions should not be instanced by the user. Intrinsic functions are documented in this section.

A user-written function is a function written by a PS 390 user for a specific application. That application may perform operations not provided by the PS 390 intrinsic functions or perform operations that would require a large network of intrinsic functions to accomplish. User-written functions are documented in Section *AP5*.

Functions must be "instanced" before they can be incorporated into a function network. Instancing is the process of creating a unique case of the function. The unique case is identified by the user- or system-given name, and the input and output connections of the function.

1. Intrinsic Functions

Intrinsic functions are the master set of function "templates" which are instanced and used in building function networks. These functions are of the form

F:identifier

where "identifier" is the name of the function (e.g., ROUTE, MUL, CONCATENATE). Using the **Name := F:identifier;** command, the user can create uniquely named instances of intrinsic functions.

For example,

```
Adder := F:ADD;
```

creates a function called **Adder**, which is a uniquely named instance of the F:ADD intrinsic function. Inputs and outputs of user-instanced functions are connected to create function networks for handling data input from the interactive devices, from the host computer, or from other functions. For example,

```
CONNECT Adder<1>:<1>Multiply;
```

connects output 1 of the function instance **Adder** to input queue 1 of the function instance **Multiply**.

Whenever the PS 390 is booted, certain intrinsic functions are automatically instanced for use, and are called initial function instances. Initial function instances are documented in Section *RM3*.

Intrinsic functions are documented in this section. Intrinsic user functions and listed first, followed by the intrinsic system functions. Appendix A contains a listing of the intrinsic user and system functions by category.

Since some functions use the ASCII decimal equivalent of characters, an ASCII chart with decimal codes is included after Appendix A.

Unless noted, all strings consist of 8-bit ASCII characters.

2. Function Representation

Functions are represented as "black boxes" with numbered inputs and outputs enclosed in angle brackets. Valid data types are shown in abbreviated form at each input and output. A "C" in the function name usually indicates that one or more input queues contain a constant value. A constant input is shown by the letter "C" following the input number in angle brackets. The following is a key to the abbreviations used.

KEY TO VALID DATA TYPES	
Any	Any message
B	Boolean value
C	Constant vlaue
CH	Character
I	Integer
Label	Data input to LABELS node
M	2x2, 3x3, 4x3, 4x4 matrix
PL	Pick list
R	Real number
S	Any string
Special	Special data type
V	Any vector
2D	2D vector
3D	3D vector
4D	4D vector
2x2	2x2 matrix
3x3	3x3 matrix
4x3	4x3 matrix
4x4	4x4 matrix

3. Conjunctive/Disjunctive Sets

Some PS 390 functions have conjunctive or disjunctive inputs and outputs. A function with conjunctive inputs must have a new message on every input before it will fire. A function with conjunctive outputs will send a message on every output when the function is fired.

Conversely, a disjunctive input function does not require a new message on every input to fire. A disjunctive output function may not send a message on each output (or any output) every time it receives a complete set of input messages.

The F:ADD function, for example, has conjunctive inputs. A value must be sent to each of the two inputs before the function will fire. The inputs are then added together, which produces an output that is the sum of the inputs. The output is conjunctive. Unlike F:ADD, F:ADDC is a disjunctive input function; it does not require a new message on every input.

F:BROUTE, on the other hand, is a conjunctive input, disjunctive output function. Both inputs require messages to fire the function. However, a message will be sent out only one of the outputs, depending on the value received on input 1.

F:ACCUMULATE is an example of a different sort of disjunctive output. Every input does not produce an output. The function activates each time a new message is received on input 1, but the output fires at specified intervals rather than each time the function is activated.

The following notation is used in to indicate disjunctive or conjunctive inputs and outputs.

KEY TO CONJUNCTIVE/DISJUNCTIVE SYMBOLS	
CC	conjunctive inputs, conjunctive outputs
CD	conjunctive inputs, disjunctive outputs
DC	disjunctive inputs, conjunctive outputs
DD	disjunctive inputs, disjunctive outputs

4. Constant and Active Queues

Function input queues are of two different types:

1. Constant queues, where the queue retains a message until another message is received on that input, and any previous message on that queue is removed. Constant queues cannot be emptied unless the data on the queue is of an inappropriate type. The message is not "used up" by the function.
2. Active queues (sometimes called trigger queues), where all data are collected on the queue, and then input to the function on a "first in first out" basis as soon as the function is activated. Messages on active queues are "used up" by the function.

A function usually requires something to be on every input queue before it can execute.

The programmer has the ability to change the type of input queues on a function. The following PS 390 command:

```
SETUP CNESS TRUE <n>Name;      {makes it a constant queue}
SETUP CNESS FALSE <n>Name;     {makes it an active queue}
```

will set the *nth* queue of the function Name to be a Constant queue if TRUE is entered, and to be a Active queue if FALSE is entered. Unless specified otherwise, input queues are by default Active queues.

This feature should be used only when a function is first instanced. Input queues should not be changed between active and constant at any time after the function has started processing data.

This feature is accessible for most user instanceable functions. Functions which specify their queue characteristics by their name (i.e. F:ADDC) will continue to be instanced with the same defaults as before. There are a few functions which, because of their nature, are not allowed to change queue characteristics. These functions are:

- F:BOOLEAN_CHOOSE
- F:CI(n)
- F:CLCSECONDS
- F:CLFRAMES
- F:CLTICKS
- F:GATHER_GENFCN
- F:INPUTS_CHOOSE(n)
- F:K2ANSI
- F:LINEEDITOR
- F:LIST
- F:PICK
- F:RASTER
- F:TEDUP
- F:VT10

When this command is applied to one of these functions, the following error message is given:

```
E 102 *** Cannot affect Cness for its generic function: (Name)
```

When this command is applied to a name that is not a function instance the following error message is given:

```
E 95 *** Name must be a function instance
```

The user should exercise caution when setting Cness. For example, if all input queues were set to constant, the function would be constantly firing. Changing Cness in one of the functions named above will have no effect or may cause the function to work in an unpredictable manner. The user is advised to exercise caution when setting Cness for other functions.

5. Queue Data Types

Blocks of data passed between functions are referred to as Qdata message blocks. The names and definitions of the general data types acceptable by function input queues are given in the following table:

Qdata Type	DEFINITION
Qreset	Dataless: reset a function instance
Qprompt	Dataless: Flush the CI pipeline
Qboolean	Normal carrier of boolean values
Qinteger	Normal carrier of integer values
Qreal	Normal carrier of floating point values
Qpacket	Normal carrier of byte strings
Qmorepacket	Alternate to Qpacket as carrier of byte string on input to PS 390 (only occurs as output from F:DEPACKET, F:CIROUTE)
Qmove2	2D vector including P bit
Qdraw2	2D vector including the L bit
Qvec2	2D vector with no P/L bit (normal vector)
Qmove3	3D vector including P bit
Qdraw3	3D vector including the L bit
Qvec3	3D vector with no P/L bit (normal vector)
Qmove4	4D vector including P bit
Qdraw4	4D vector including the L bit
Qvec4	4D vector with no P/L bit (normal vector)
Qmat2	2x2 matrix (all matrices use 4x4 indexing)
Qmat3	3x3 matrix
Qmat4	4x4 matrix

6. Intrinsic User Functions

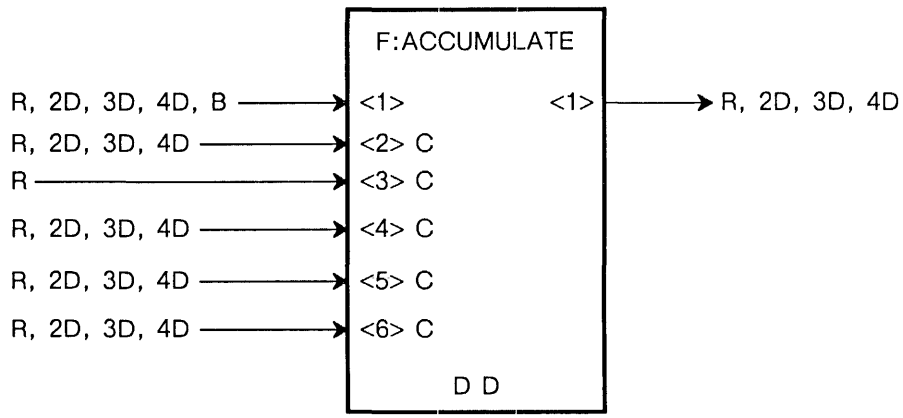
Following is a summary of the Intrinsic User Functions. The functions are ordered alphabetically on a letter-by-letter basis.

The following information, where relevant, is given for each function:

- Name
- Type/Category
- Purpose
- Description of inputs and outputs
- Defaults
- Notes
- Associated functions
- Examples

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accumulates a series of input values and sends the sum at specified intervals.

DESCRIPTION

INPUTS

<1> — value to be accumulated
<2> — initial value (constant)
<3> — output interval (constant)
<4> — scale factor (constant)
<5> — upper limit on sum (constant)
<6> — lower limit on sum (constant)

OUTPUT

<1> — sum

DEFAULTS

Input <3> defaults to 0, input <4> defaults to 1.

NOTES

1. The input values may be scaled, and the output values may be limited to a specified range as in F:LIMIT. Note that this combination of operations is especially useful for handling input from the control dials.

2. An initial value must be sent to input <2>; subsequent values are sent to input <1>. All values at input <1> are scaled by input <4> before adding.

3. The sum is output whenever it differs from the previous F:ACCUMULATE output (or zero if there was no previous output) by more than the value at input <3>. (If vectors are being accumulated, this difference and the value at input <3> are taken to be vector lengths and, therefore, real numbers. Vector lengths are considered to be $n(x,y) = |x|+|y|$, not $n(x,y) = x^2 + y^2$.

4. Inputs <5> and <6> specify limits (upper and lower, respectively) to be applied to the accumulated sum. A sum falling outside the range is adjusted to the nearer limit, and any further accumulations operate on the limited sum.

5. Inputs <1> and <2> must be of the same data type. To change the data type of the sum to be accumulated, send a new initial value of the appropriate type to <2>. Note that the data type of the accumulated sum may not be changed simply by starting to send different data types to <1>—these will only generate an "Incompatible inputs" error message.

6. If input <2> is a real number, then inputs <4>, <5>, and <6> must be real numbers. On the other hand, if input <2> is a vector, then each of inputs <4>, <5>, and <6> may be either a vector of the same dimension as <2> or a real number.

7. If vectors are being accumulated, but the scale factor at <4> is real, then each coordinate of each vector accumulated at <1> is multiplied by the real scale factor before the vector is added in. If the scale factor at <4> is a vector, each of its coordinates is multiplied by the corresponding coordinate of the accumulated vector.

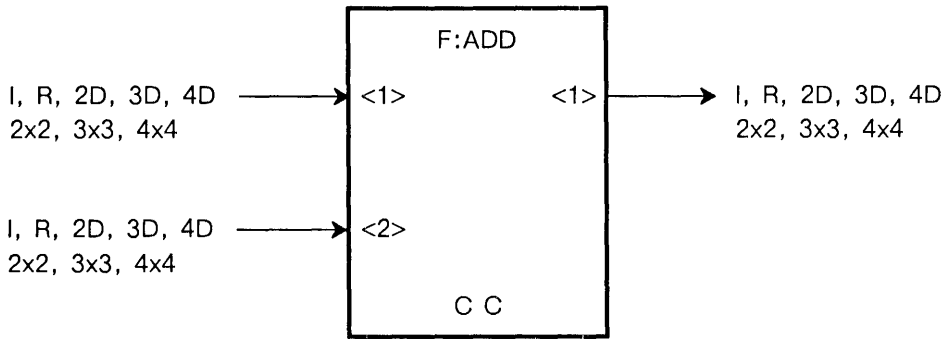
8. If vectors are being accumulated, but both the upper sum limit at <5> and the lower sum limit at <6> are real, then these real numbers are the limits for each coordinate of the sum. If <5> and <6> are vectors, each of their respective coordinates is applied as a limit to the corresponding coordinate of the sum.
9. If input <1> is Boolean (regardless of value), the current sum is immediately sent to output <1>. If you send a new value to input <2> and then send a Boolean value to input <1>, the accumulator will be reset to the new value and this value is immediately sent to output <1>.
10. Vector types may not be mixed in an F:ACCUMULATE operation; all vectors must be either 2D, 3D, or 4D.

EXAMPLE

Refer to Application Note 10 in Section *TT1*.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs and produces an output that is the sum of those inputs.

DESCRIPTION

INPUTS

<1> — input value

<2> — input value

OUTPUT

<1> — sum

NOTE

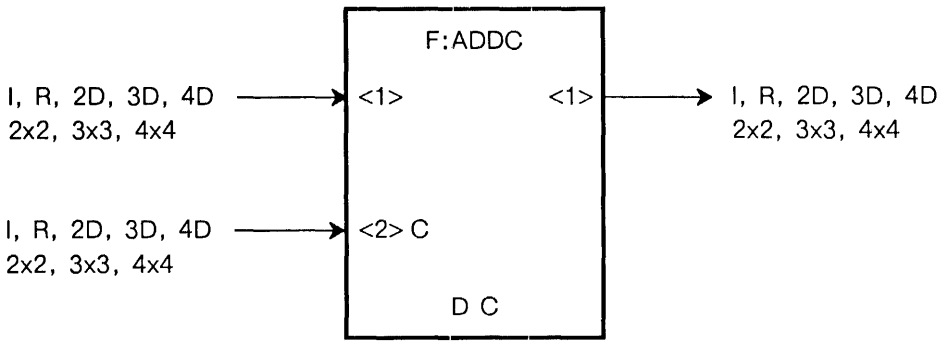
The two input values must be of the same data type (except that a combination of a real number and an integer is allowed); the output data type depends on the input data type(s). If an integer is added to a real number the output is a real number.

ASSOCIATED FUNCTION

F:ADDC

TYPE

Intrinsic User Function — Arithmetical and Logical



PURPOSE

Accepts two inputs and produces an output that is the sum of those inputs. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — input value

<2> — input value (constant)

OUTPUT

<1> — sum

NOTE

The two input values must be of the same data type (except that a combination of a real number and an integer is allowed); the output data type depends on the input data type(s). If an integer is added to a real number the output is a real number.

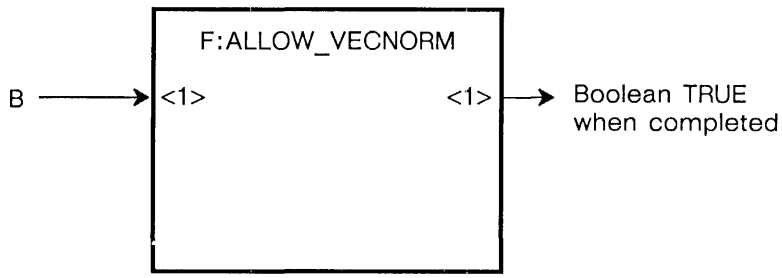
ASSOCIATED FUNCTION

F:ADD

F:ALLOW_VECNORM

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

F:ALLOW_VECNORM allows vector-normalized vector lists to be created locally or downloaded from the host to the PS 390. Enhanced CPK firmware is dependent on vector-normalized data to perform renderings.

DESCRIPTION

INPUT

<1> — Boolean value

OUTPUT

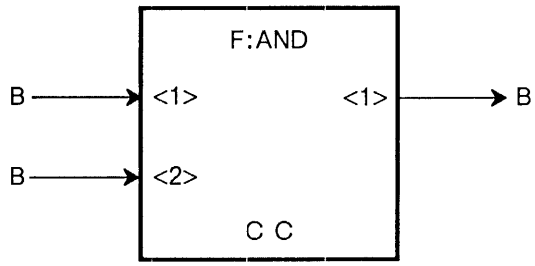
<1> — Boolean TRUE when completed

NOTE

A Boolean TRUE sent to input <1> of F:ALLOW_VECNORM allows vector-normalized data to be created by the PS 390. A Boolean FALSE sent on input <1> will reset the PS 390 and cause vector-normalized data to be converted to block-normalized data. A Boolean TRUE is sent from output <1> when the function has run to completion.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two Boolean values as input and produces a Boolean value output that is the logical AND of the two inputs.

DESCRIPTION

INPUTS

<1> — Boolean value input

<2> — Boolean value input

OUTPUT

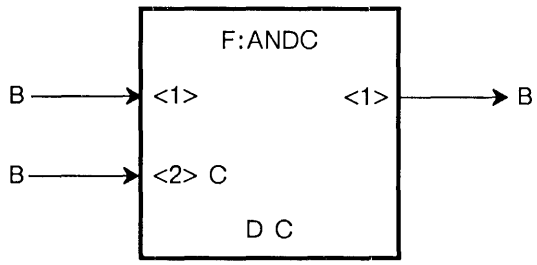
<1> — logical AND of the two inputs

ASSOCIATED FUNCTION

F:ANDC

TYPE

Intrinsic User Function — Arithmetical and Logical



PURPOSE

Accepts two Boolean values as input and produces a Boolean value output that is the logical AND of the two inputs. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — Boolean value input

<2> — Boolean value input (constant)

OUTPUT

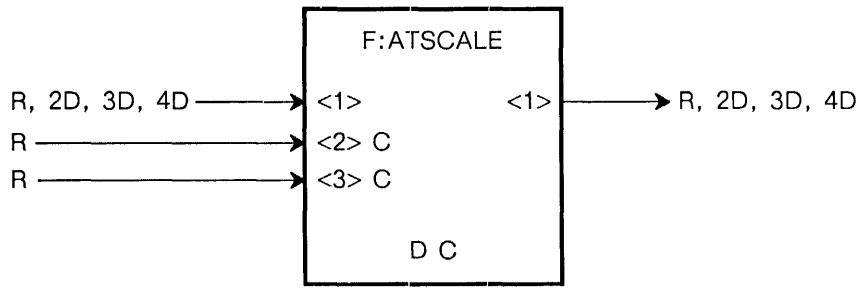
<1> — logical AND of the two inputs

ASSOCIATED FUNCTION

F:AND

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Like F:ACCUMULATE, F:ATSCALE accumulates the sum of a series of real numbers or vectors. Unlike F:ACCUMULATE, its sum is cleared after output.

DESCRIPTION

INPUTS

<1> — value to be accumulated
<2> — scale factor (constant)
<3> — delta (constant)

OUTPUT

<1> — accumulated sum

DEFAULTS

Input <2> = 1.0, Input <3> = 0.0

NOTES

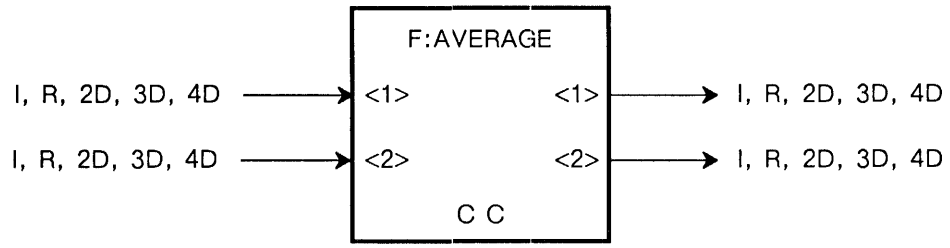
1. Each value on input <1> is scaled by the value on input <2>, then added to the internally stored current sum of scaled input <1> values. When the accumulated sum differs from the last value sent out output <1> by at least the amount on input <3>, the accumulated sum is output and the internal accumulated sum is cleared.

2. If vectors are input on <1>, the difference on input <3> is taken to be vector length. Vector length is the linear distance from a vector location to the origin of the world coordinate system (i.e., the Euclidean norm, $n(x,y) = x^2 + y^2$).

3. Sending a Boolean TRUE or FALSE to input <1> forces the accumulated sum to be output and cleared from internal storage.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs, outputs the average of the two inputs on output <1>, and outputs the value of input <2> unchanged on output <2>.

DESCRIPTION

INPUTS

<1> — any value

<2> — any value

OUTPUTS

<1> — average of the two input values

<2> — value of input <2> unchanged

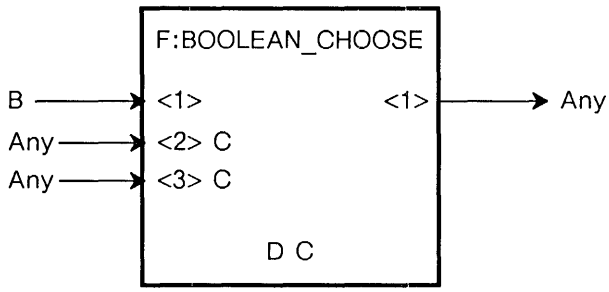
NOTE

The two input values must be of the same data type (except that a combination of a real number and an integer is allowed); the outputs are also of that data type. If an integer is averaged with a real number, a real number is output on <1>.

F:BOOLEAN_CHOOSE

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Uses the Boolean value on input <1> to select the constant message on input <2> or input <3>, outputting the selected message on output <1>.

DESCRIPTION

INPUTS

<1> — Boolean value

<2> — any message (constant)

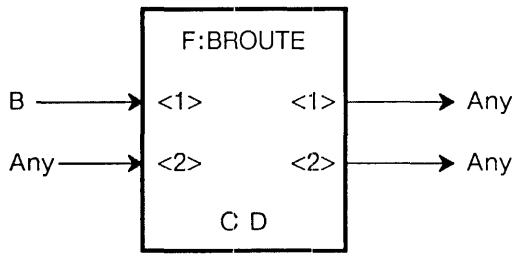
<3> — any message (constant)

OUTPUT

<1> — message on input <2> when input <1> is TRUE or message on input <3> when input <1> is FALSE

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Acts as a Boolean routing function, accepting a Boolean value on input <1> and any message on input <2>. When a TRUE is received on input <1>, the message appears at output <1>. When a FALSE is received on input <1>, the message appears at output <2>.

DESCRIPTION

INPUTS

<1> — trigger

<2> — any message

OUTPUTS

<1> — message on input <2> when input <1> is TRUE

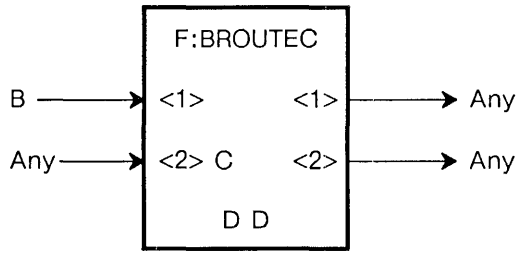
<2> — message on input <2> when input <1> is FALSE

ASSOCIATED FUNCTIONS

F:BROUTEC, F:CBROUTE

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Acts as a Boolean routing function, accepting a Boolean value on input <1> and any message on constant input <2>. When a TRUE is received on input <1>, the message appears at output <1>. When a FALSE is received on input <1>, the message appears at output <2>.

DESCRIPTION

INPUTS

<1> — trigger

<2> — any message (constant)

OUTPUTS

<1> — message on input <2> when input <1> is TRUE

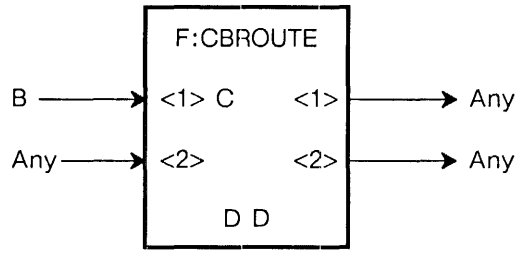
<2> — message on input <2> when input <1> is FALSE

ASSOCIATED FUNCTIONS

F:BROUTE, F:CBROUTE

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Acts as Boolean routing function, sending the message on input <2> to output <1> when the constant Boolean value on input <1> is TRUE or to output <2> when the constant Boolean value on input <1> is FALSE.

DESCRIPTION

INPUTS

<1> — trigger (constant)

<2> — any message

OUTPUTS

<1> — message on input <2> when input <1> is TRUE

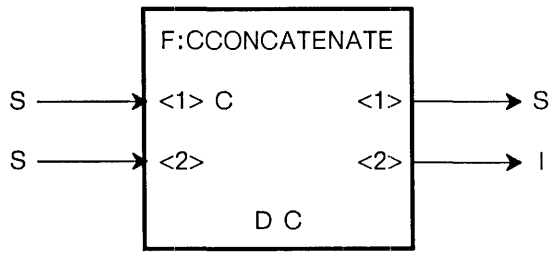
<2> — message on input <2> when input <1> is FALSE

ASSOCIATED FUNCTIONS

F:BROUTE, F:BROUTEC

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts two ASCII character strings and outputs on output <1> a string that is formed by concatenating the string on input <2> behind the string on input <1>. The length of the resulting string is sent on output <2>. Input <1> is a constant.

DESCRIPTION

INPUTS

<1> — ASCII string (constant)

<2> — ASCII string

OUTPUTS

<1> — concatenated string

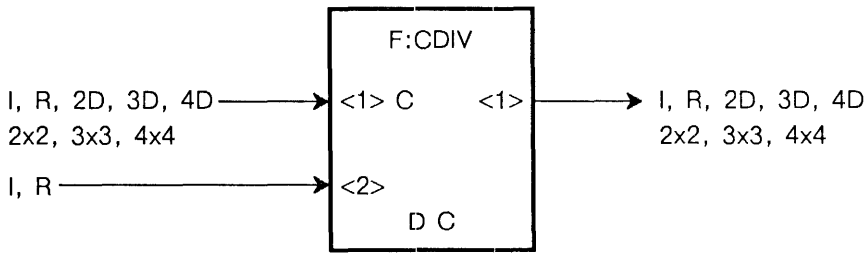
<2> — length of the concatenated string

ASSOCIATED FUNCTIONS

F:CONCATENATE, F:CONCATENATEC

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs and produces an output that is the quotient of the two inputs (input <1> divided by input <2>). Input <1> is a constant.

DESCRIPTION

INPUTS

<1> — dividend (constant)

<2> — divisor

OUTPUT

<1> — quotient

NOTE

The output is the same data type as input <1> (except when <1> is an integer and input <2> is a real number; then a real number is output). Input <2> should not be 0.

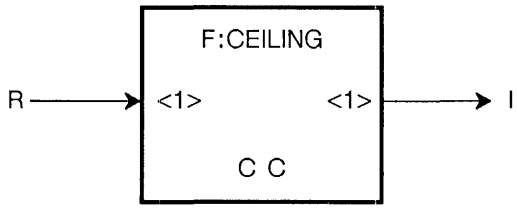
ASSOCIATED FUNCTIONS

F:DIV, F:DIVC

F:CEILING

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Rounds a real number away from zero to the nearest integer.

DESCRIPTION

INPUT

<1> — real number to be rounded

OUTPUT

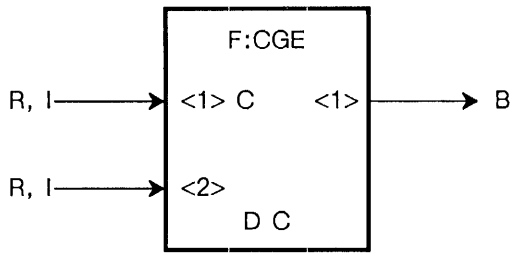
<1> — nearest integer

ASSOCIATED FUNCTION

F:FIX

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers at its two inputs and produces a Boolean value output that is TRUE if input <1> is greater than or equal to input <2>, and FALSE otherwise. Input <1> is a constant.

DESCRIPTION

INPUTS

<1> — real number or integer to be compared (constant)

<2> — real number or integer to be compared

OUTPUT

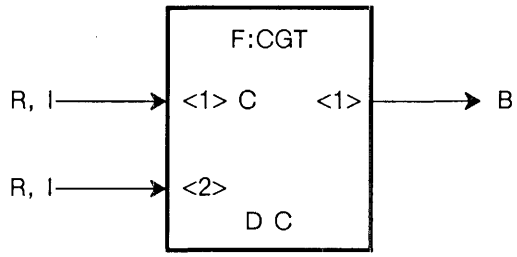
<1> — Boolean value

ASSOCIATED FUNCTIONS

F:GE, F:GEC

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of two real numbers or integers at its inputs and produces a Boolean value output that is TRUE if input <1> is greater than input <2>, and FALSE otherwise. Input <1> is a constant.

DESCRIPTION

INPUTS

<1> — real number or integer to be compared (constant)

<2> — real number or integer to be compared

OUTPUT

<1> — Boolean value

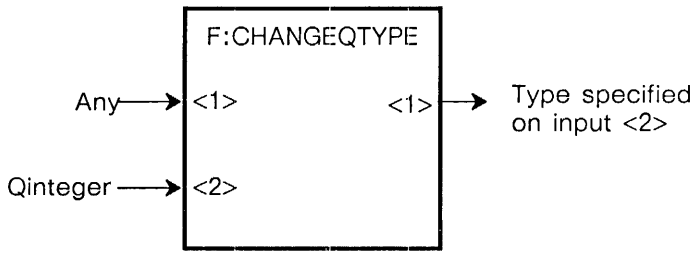
ASSOCIATED FUNCTIONS

F:GT, F:GTC

F:CHANGEQTYPE

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

This function receives messages of any type on input <1> and sends out Qdata messages with the type field changed to the type specified by the ORD of the integer on input <2>.

DESCRIPTION

INPUTS

<1> — Any

<2> — Qinteger

OUTPUT

<1> — Type specified on input <2>

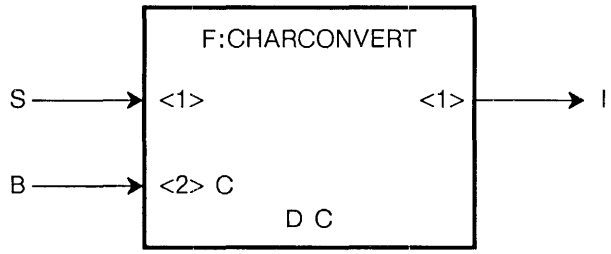
NOTE

This function should be used with extreme caution. Changing the type field will cause the system to treat it like the new type. It is a good rule to never change to or from message types which have pointer fields included in them. It is possible to change types between the Qdata types listed in section 5.

F:CHARCONVERT

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Converts the bytes of the string on input <1> into a stream of integers, one integer per byte.

DESCRIPTION

INPUTS

<1> — any string

<2> — Boolean value (constant)

OUTPUT

<1> — stream of integers

DEFAULT

Boolean TRUE on input <2>.

NOTES

1. The condition of the Boolean value determines the range of bytes as integers as follows:

   TRUE = 0 to 255
   FALSE = -128 to 127 (2's complement)

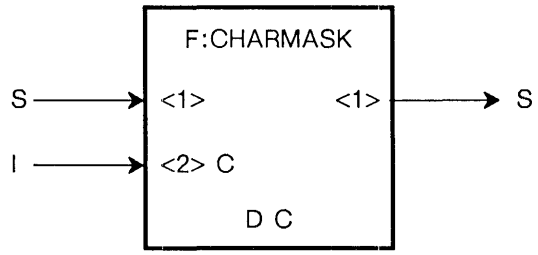
2. Note that if a TRUE is on input <2>, a value from 0–255 is output on <1>. If a FALSE is on input <2> and the value on input <1> is from 0–127, the value output is the same value that was input on <1>. If a FALSE is on input <2> and the value on input <1> is 128–255, a corresponding value between –128 and –1 is output.

EXAMPLE

'A' becomes 65
'AB' becomes 65 followed by 66

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Masks each of the bytes of the string on input <1> by ANDing it with the integer on the constant input <2>, then outputs the masked string.

DESCRIPTION

INPUTS

<1> — any string

<2> — integer (constant)

OUTPUT

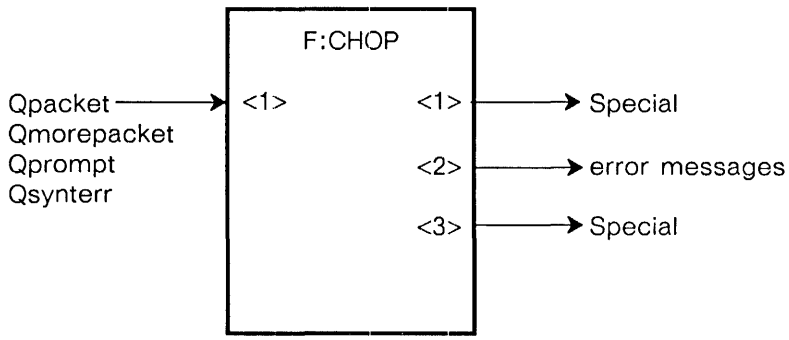
<1> — masked string

NOTE

Only the low-order byte of the integer is used in the mask, i.e., integer 256 would be a 0 mask. Therefore, numbers between 0–255 are recommended.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

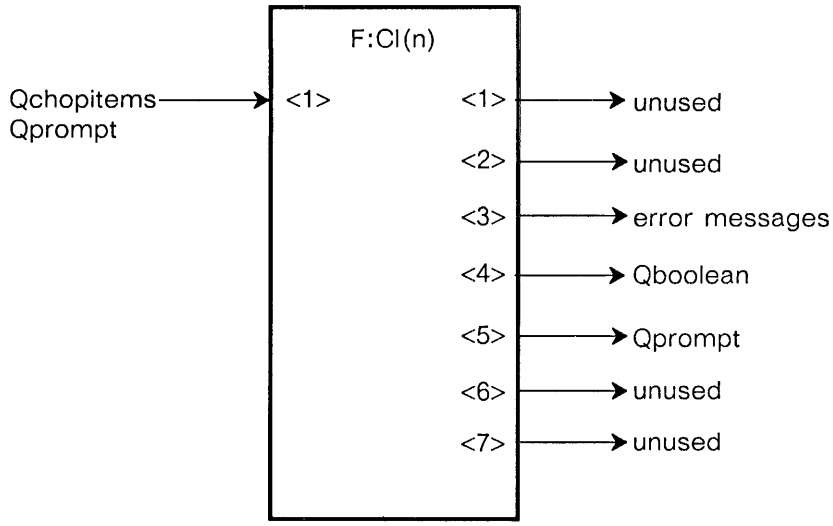
This function chops and parses the input command language generating proper messages for an instance of the function F:CI(n).

NOTES

1. Output <1> outputs special data which goes to F:CI(n) function, which is the only function that can accept this type of data.
2. Output <3> outputs special data that directs the printing of syntax error messages.

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

This function interprets commands, creating display structures and function networks. It receives input either from a chop/parse function or a READSTREAM function (if using the GSRs).

A single parameter is given when this function is instanced (for example H_CI0:=F:CI(4);). This parameter is the "CINUM" and is used to identify all names and connections this CI makes. When the CI receives an INIT command, it destroys only those connections it has made and only those structures associated with the names which have its CINUM.

NOTES

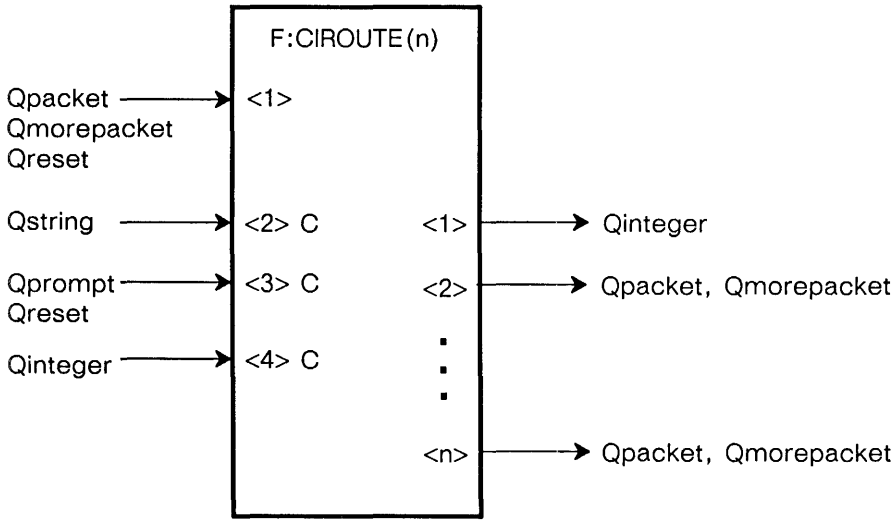
1. A name is created when that name is referenced for the first time, even if it has no associated structure. The CI that created the name is the "owner" of that name, even if the entity it refers to is created by another CI.

2. Each function has an output <0> that is used to send error messages (such as illegal input error messages). The connection from this output is made automatically by the CI that creates the function. The CI finds the appropriate error function to connect output <0> to by looking on its own output <3>.

3. Output <4> sends out a Qboolean with a TRUE value when an INIT command is entered. This output is connected to the initial function instance CLEAR_LABELS to clear out the labels on the keyboard and dials.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

F:CIROUTE(n) demultiplexes a stream of Qpackets/Qmorepackets from input <1> to one of the n output channels. The first byte of an incoming Qpacket is assumed to be the multiplexing byte - equal to the base character (from input <2>) + K, where K is the channel number. If K > (n–3) or K < 0, there is no channel for this output and a pair of messages are sent on outputs <1> and <2>. These can be used to allow for later remultiplexing or further demultiplexing. An integer giving the indicated output port is sent on output <1> and the message for which there was no defined output is sent on output <2>. Whether or not K is within the limits implied by the number of outputs of F:CIROUTE(n), the multiplexing byte is removed from the start of the packet.

F:CIROUTE(n) passes incoming Qmorepackets out the current channel (as defined by the last Qpacket). Initially, after a Qreset is received, the current channel is –1.

When instancing this function, a parameter is required to specify the number of outputs.

F:CIROUTE(n) is a special version of F:DEMUX. It assumes that it is driving parallel, asynchronous paths to a common destination (the Command Interpreter). It synchronizes those paths by sending out a Qprompt on a channel at the end of using that channel and waiting for it to come back around before switching to the next channel. This assumes that the common destination can strip Qprompts and send them back (which the CI does). Input <4> gives the maximum channel number, m, for which path flushing is desired. F:CIROUTE(n) flushes channels 0<=K <= m with Qprompts.

DESCRIPTION

INPUTS

<1> — Qpacket - switch multiplexing channel & send
Qmorepacket - send on current channel
Qreset - re-init & purge queue 3.

<2> — Qstring - base character of multiplexing byte (constant)

<3> — Qprompt (back from the CI) (constant)
Qreset - acts like Qprompt (constant)

<4> — Qinteger - max channel # to get prompts (default 0) must be an individual output (not <1>,<2>) (constant)

OUTPUTS

<1> — Qinteger - (i) when output port <3+i> doesn't exist

<2> — Qpacket, Qmorepacket - stream which didn't have any valid destination

<3+i> — Qpacket, Qmorepacket - output stream (i) where destination was given as [base-char + (i)] (possibly) Qprompt (if i <= [input <4>])

NOTE

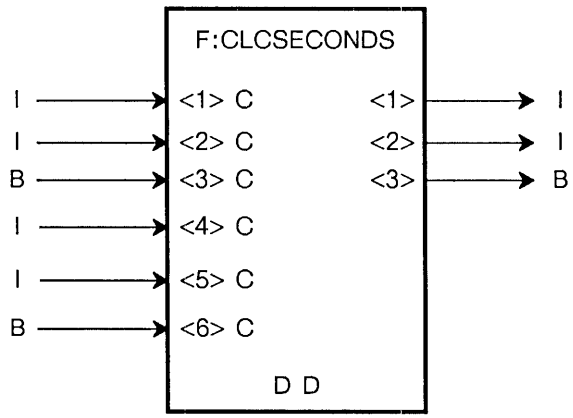
Because of the synchronizing nature of CIROUTE, it should be hooked to its complete destination network before its input <1> is connected.

EXAMPLES

Refer to Helpful Hints 7 and 9 in Section *TT2*.

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

Generates outputs at timed intervals as specified by the inputs. All inputs to F:CLCSECONDS are constants. All outputs occur at the same timed interval. (Output <1> may be disabled.)

DESCRIPTION

INPUTS

<1> — timed interval (constant)
<2> — number of time intervals (constant)
<3> — gate (constant)
<4> — integer A (constant)
<5> — integer B (constant)
<6> — TRUE = run, FALSE = stop (constant)

OUTPUTS

<1> — integer A+B if input <3> is TRUE
<2> — integer A+B
<3> — TRUE if input <2> is not exceeded

NOTES

1. Input <1> is an integer that specifies a timed interval in hundredths of a second. Outputs from the function occur at this interval. Thus, a 10 on input <1> would specify a time interval of 1/10 second.
2. Input <2> is an integer that specifies the number of time intervals (duration) that the Boolean value on output <3> will be TRUE. When this number of intervals is exceeded, the Boolean value will be output as FALSE on each succeeding interval. Input <2> may be reset at any time, since the value at this input is decremented by 1 with each execution.
3. Input <3> is a Boolean value that is used to gate the integer on output <1>. If the Boolean value is TRUE, the integer (A+B) is output at each timed interval. If the Boolean value is FALSE, output <1> is disabled.
4. Inputs <4> and <5> are integers A and B, respectively. The sum of these integers is output as an integer on output <1> if the Boolean value on input <3> is TRUE. This sum (A+B) is output as an integer on output <2>, independent of the condition of the Boolean value on input <3>.
5. Input <6> is an optional switch. If input <6> receives no messages, the timer will run when there is a message on all of inputs <1> through <5>. If a Boolean FALSE is received on input <6>, the timer waits for a Boolean TRUE to be received on input <6> before running. No outputs are generated so long as input <6> is FALSE.

ASSOCIATED FUNCTIONS

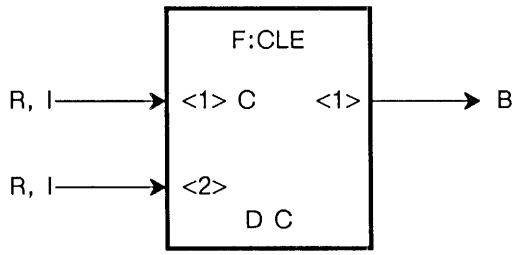
F:CLFRAMES, F:CLTICKS

EXAMPLE

Refer to Application Notes 11 and 12 in Section *TTI*.

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers or integers at its inputs, and produces a Boolean value output that is TRUE if input <1> is less than or equal to input <2> and FALSE otherwise. Input <1> is a constant.

DESCRIPTION

INPUTS

<1> — value to be compared (constant)

<2> — value to be compared

OUTPUT

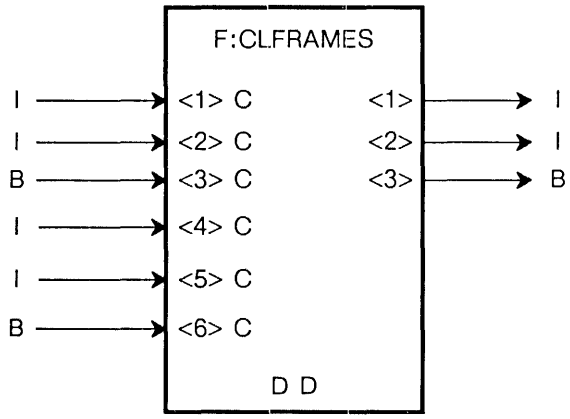
<1> — Boolean value

ASSOCIATED FUNCTIONS

F:LE, F:LEC

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

Identical to F:CLCSECONDS and F:CLTICKS, except that the time source is refresh frames.

DESCRIPTION

INPUTS

<1> — timed interval (constant)
<2> — number of time intervals (constant)
<3> — gate (constant)
<4> — integer A (constant)
<5> — integer B (constant)
<6> — TRUE = run, FALSE = stop (constant)

OUTPUTS

<1> — A+B if input <3> is TRUE
<2> — A+B
<3> — TRUE if input <2> is not exceeded

NOTES

1. Input <1> is an integer that specifies a timed interval in frames. A frame is the length of time the display processor takes to draw the current structure once. The refresh rate is the number of frames per second. Outputs from the function occur at this interval.

2. Input <2> is an integer that specifies the number of timed intervals (duration) that the Boolean value on output <3> will be TRUE. When this number of intervals is exceeded, the Boolean value will be output as FALSE on each succeeding interval. Input <2> may be reset at any time.

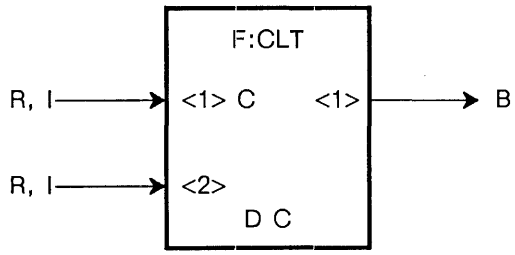
3. Input <3> is a Boolean value that is used to gate the integer on output <1>. If the Boolean value is TRUE, the integer (A+B) is output each timed interval. If the Boolean value is FALSE, output <1> is disabled.

4. Inputs <4> and <5> are integers A and B, respectively. The sum of these integers is output as an integer on output <1> if the Boolean value on input <3> is TRUE. This sum (A+B) is output as an integer on output <2>, independent of the condition of the Boolean value on input <3>.

5. Input <6> is an optional switch. If input <6> receives no messages, the timer will run when there is a message on all of inputs <1> through <5>. If a Boolean FALSE is received on input <6>, the timer waits for a Boolean TRUE to be received on input <6> before running. No outputs are generated so long as input <6> is FALSE.

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers or integers at its inputs, and produces a Boolean value output that is TRUE if input <1> is less than input <2>, and FALSE otherwise. Input <1> is a constant.

DESCRIPTION

INPUTS

<1> — value to be compared (constant)

<2> — value to be compared

OUTPUT

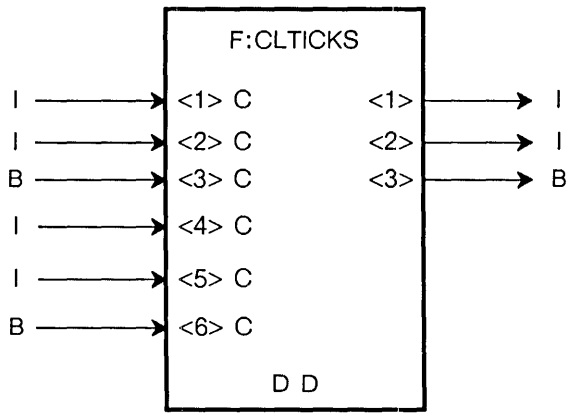
<1> — Boolean value

ASSOCIATED FUNCTIONS

F:LT, F:LTC

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

Identical to F:CLCSECONDS and F:CLFRAMES, except that the time source is ticks of the 20 Hz system clock.

DESCRIPTION

INPUTS

<1> — timed interval (constant)
<2> — number of time intervals (constant)
<3> — gate (constant)
<4> — integer A (constant)
<5> — integer B (constant)
<6> — TRUE = run, FALSE = stop (constant)

OUTPUTS

<1> — A+B if input <3> is TRUE
<2> — A+B
<3> — TRUE if input <2> is not exceeded

NOTES

1. Input <1> is an integer that specifies a timed interval in ticks (where a tick is half the duration of the alternating current supply, 1/20 second in the U.S.). Outputs from the function occur at this interval.

2. Input <2> is an integer that specifies the number of timed intervals (duration) that the Boolean value on output <3> will be TRUE. When this number of intervals is exceeded, the Boolean value will be output as FALSE on each succeeding interval. Input <2> may be reset at any time.

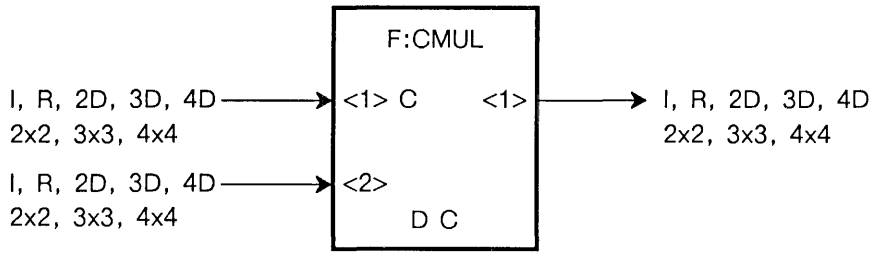
3. Input <3> is a Boolean value that is used to gate the integer output <1>. If the Boolean is TRUE, the integer (A+B) is output each timed interval. If the Boolean is FALSE, output <1> is disabled.

4. Inputs <4> and <5> are integers A and B, respectively. The sum of these integers is output as an integer on output <1> if the Boolean value on input <3> is TRUE. This sum (A+B) is output as an integer on output <2>, independent of the condition of the Boolean value on input <3>.

5. Input <6> is an optional switch. If input <6> receives no messages, the timer will run when there is a message on all of inputs <1> through <5>. If a Boolean FALSE is received on input <6>, the timer waits for a Boolean TRUE to be received on input <6> before running. No outputs are generated so long as input <6> is FALSE.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs and outputs the product of the two inputs. Input <1> is a constant.

DESCRIPTION

INPUTS

<1> — multiplier (constant)

<2> — multiplicand

OUTPUT

<1> — product

NOTE

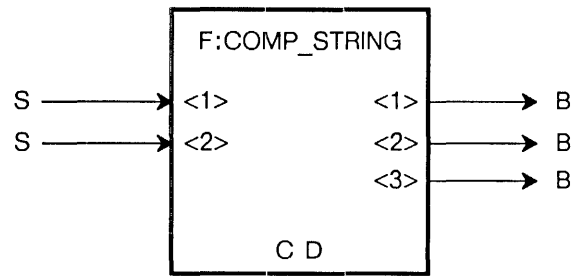
The two input values must be compatible data types; the output data type depends on the combination of input data types. Vectors are taken to be either row vectors (input <1>) or column vectors (input <2>).

ASSOCIATED FUNCTIONS

F:MUL, F:MULC

TYPE

Intrinsic User Function — Comparison



PURPOSE

Compares two strings and sends a TRUE on output <1> if string 1 is less than string 2, and a FALSE if otherwise. A TRUE is sent on output <2> if string 1 is equal to string 2, and a FALSE if otherwise. A TRUE is sent on output <3> if string 1 is greater than string 2, and a FALSE if otherwise.

DESCRIPTION

INPUTS

<1> — string

<2> — string

OUTPUTS

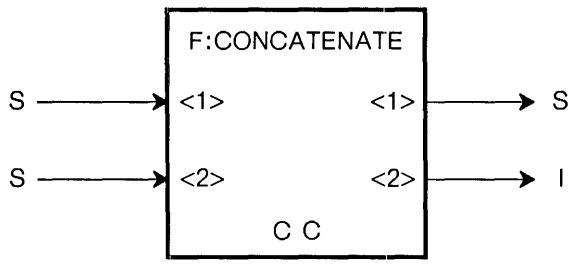
<1> — TRUE = less than

<2> — TRUE = equal to

<3> — TRUE = greater than

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts two ASCII character strings and outputs a string that is formed by concatenating the string on input <2> behind the string on input <1>. The length of the resulting string is sent on output <2>.

DESCRIPTION

INPUTS

<1> — ASCII string

<2> — ASCII string

OUTPUTS

<1> — concatenated string

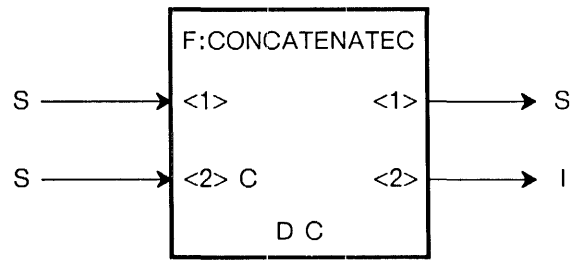
<2> — length of the concatenated string

ASSOCIATED FUNCTIONS

F:CCONCATENATE, F:CONCATENATEC

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts two ASCII character strings and outputs a string that is formed by concatenating the string on input <2> behind the string on input <1>. The length of the concatenated string is sent on output <2>. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — ASCII string

<2> — ASCII string (constant)

OUTPUTS

<1> — concatenated string

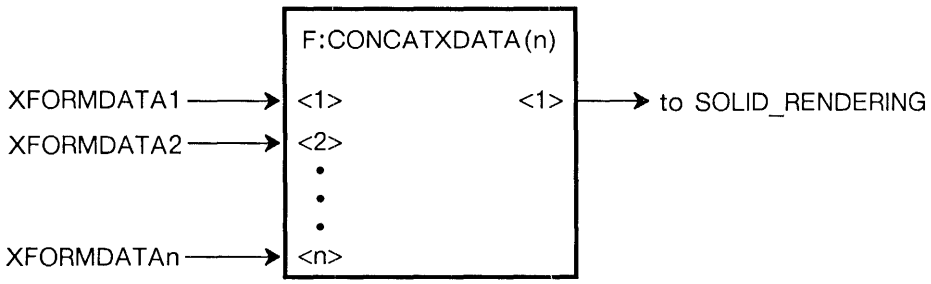
<2> — length of the concatenated string

ASSOCIATED FUNCTIONS

F:CCONCATENATE, F:CONCATENATE

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts up to 127 transformed vector lists (output from XFORMDATA functions) and concatenates them into a single transformed vector list.

DESCRIPTION

INPUTS

<1> — output of F:XFORMDATA (transformed vector list)

•
•
•

<n> — output of F:XFORMDATA (transformed vector list)

OUTPUT

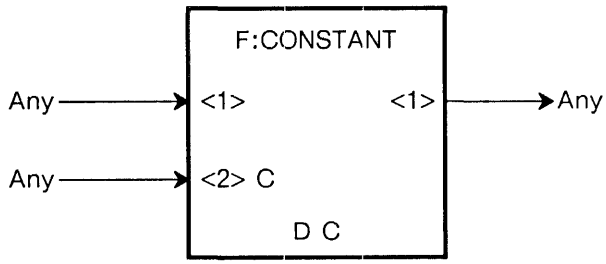
<1> — concatenated vector list

NOTES

1. This function is used to avoid the maximum vector restriction on the output of F:XFORMDATA. The XFORMDATA function will return a maximum of 2048 vectors. To obtain a rendering on the raster display of greater than 2048 vectors, the output of multiple instances of XFORMDATA must be concatenated into a single transformed vector list which can be sent to the rendering node.
2. Inputs <1> through <n> accept transformed vector lists output from F:XFORMDATA.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts any message on inputs <1> and <2>. Input <2> is a constant. The constant message on input <2> is output on <1> whenever a message is received on input <1>.

DESCRIPTION

INPUTS

<1> — trigger

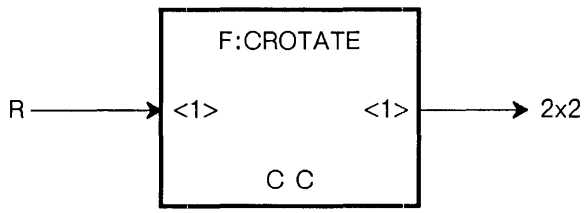
<2> — any message (constant)

OUTPUT

<1> — message on input <2> when triggered

TYPE

Intrinsic User Function — Character Transformation



PURPOSE

Creates a 2x2 Z-rotation matrix.

DESCRIPTION

INPUT

<1> — degrees of rotation in Z

OUTPUT

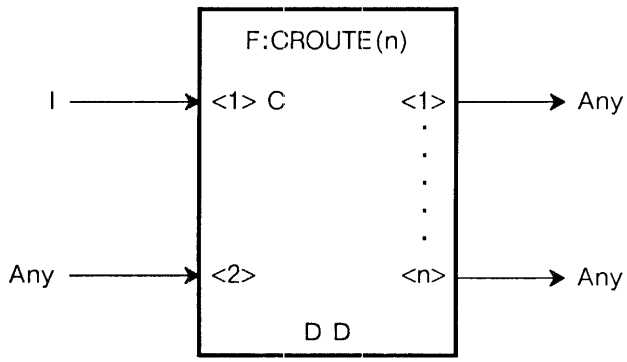
<1> — 2x2 rotation matrix

NOTES

1. The rotation matrix created by the function is normally used to update 2x2 matrix nodes in a display structure.
2. The "C" in the function's name stands for "character." 2x2 matrix nodes in display structures only affect character data nodes.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts an integer on input <1> to switch the message on input <2> to the output specified by that integer. The message on input <2> may be of any data type. The integer on input <1> is a constant.

DESCRIPTION

INPUTS

<1> — integer (valid range 1 - 127) (constant)
<2> — any message

OUTPUTS

<1> — message on input <2> when selected
•
•
•
<n> — message on input <2> when selected

NOTE

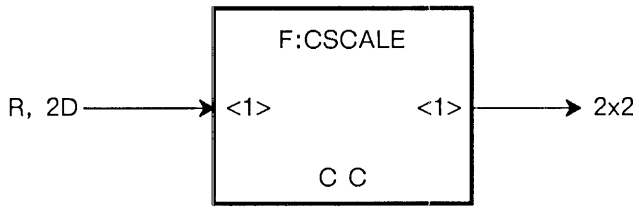
The "n" in the function name may be any integer from 2 to 127. If the integer on input <1> is not a number from 1 to n, inclusive, then an error is detected and reported.

ASSOCIATED FUNCTIONS

F:ROUTE(n), F:ROUTEC(n)

TYPE

Intrinsic User Function — Character Transformation



PURPOSE

Scales characters. Accepts a real number or a 2D vector as a scaling factor for character strings. A 2x2 scaling matrix is output.

DESCRIPTION

INPUT

<1> — scaling factor

OUTPUT

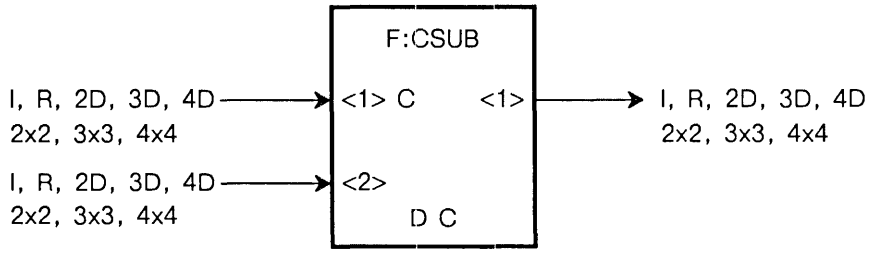
<1> — 2x2 scaling matrix

NOTES

1. The scaling matrix is normally used to update a 2x2 matrix node in a display structure. The "C" in the function's name stands for "character." Only character data nodes are affected by 2x2 matrices.
2. If a real number is input, the scaling factor represented by the real value is applied in X and Y. If a 2D vector is input, the X component of the vector is the scaling factor for X and the Y component of the vector is the scaling factor for Y.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs and produces an output that is the difference of the two inputs (input <2> is subtracted from input <1>). Input <1> is a constant.

DESCRIPTION

INPUTS

<1> — minuend (constant)

<2> — subtrahend

OUTPUT

<1> — difference

NOTE

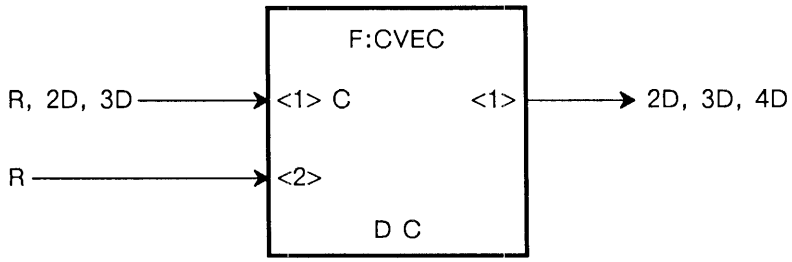
The two input values must be of the same data type (except that a combination of a real number and an integer is allowed); the output data type depends on the input data type(s).

ASSOCIATED FUNCTIONS

F:SUB, F:SUBC

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts two real numbers and outputs a 2D vector; accepts a 2D vector and a real number and outputs a 3D vector; or accepts a 3D vector and a real number and outputs a 4D vector.

DESCRIPTION

INPUTS

<1> — real number, 2D, or 3D vector (constant)

<2> — real number

OUTPUT

<1> — 2D vector if input <1> is a real number
3D vector if input <1> is a 2D vector
4D vector if input <1> is a 3D vector

NOTE

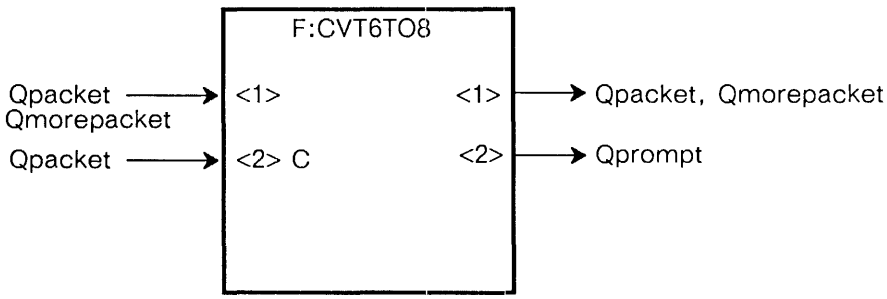
The output vector is the constant real number or vector from input <1> with the real number from input <2> appended as the last vector component.

ASSOCIATED FUNCTIONS

F:VEC, F:VECC

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

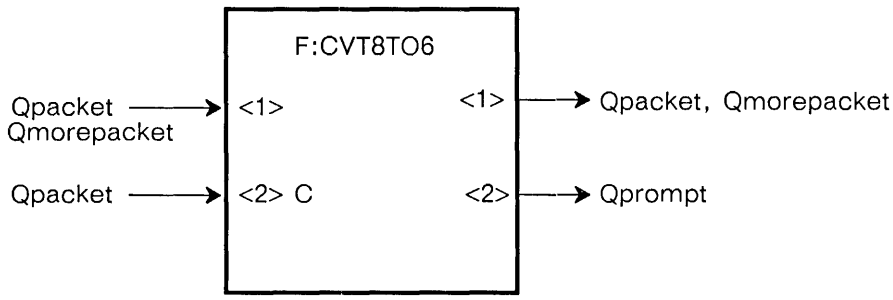
This function converts a 6-bit stream to an 8-bit stream.

Conversion is the inverse of that described in F:CVT8TO6 with the base character coming on input <2>.

Qprompts are passed out output <2>.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

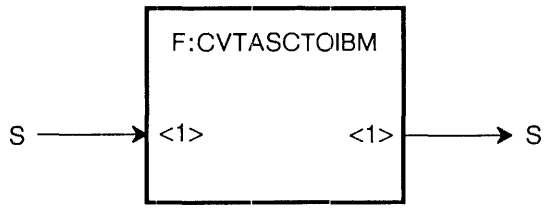
This function converts an 8-bit byte stream to a 6-bit byte stream.

The conversion yields a stream with characters from base-char (from input <2>) through base-char + 63 standing for 6-bit values in groups of 6. In addition, the special characters take care of streams that do not have a byte count = 0 mod 4. Prefixing the last group of 6 output bytes (encoded), the char: base – i: means the last i 8-bit bytes are not real.

Any Qprompts coming in on <1> are passed out output <2>.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

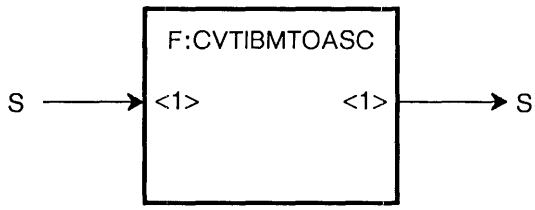
F:CVTASCTOIBM accepts packets of ASCII characters on input <1> and outputs packets of EBCDIC characters on output <1>.

EXAMPLE

Send in char(65) — get out char(193)
Send in 'AB' — get out char(193) concatenated with char(194)

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

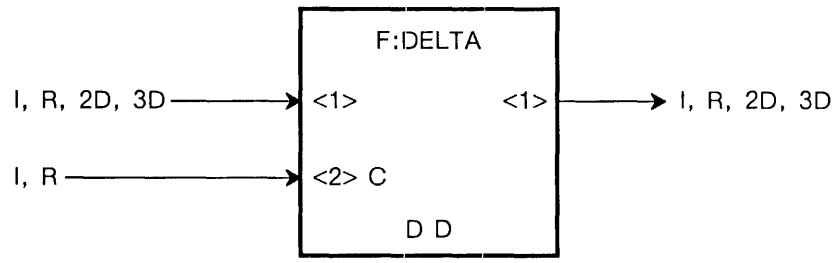
F:CVTIBMTOASC accepts packets of EBCDIC characters on input <1> and outputs packets of ASCII characters on output <1>.

EXAMPLE

Send in char(193) — get out char(65)
Send in char(193) concatenated with char(194) — get out 'AB'

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts integers, real numbers, 2D vectors, and 3D vectors on input <1> and integers or real numbers on input <2>. The value on input <1> is output on <1> if it differs in magnitude from the previous input <1> value by at least the constant delta value on input <2>.

DESCRIPTION

INPUTS

<1> — integer, real number, 2D, 3D vector

<2> — delta value (constant)

OUTPUT

<1> — value on input <1> if it differs from the previous input <1> by at least the delta value on input <2>

DEFAULT

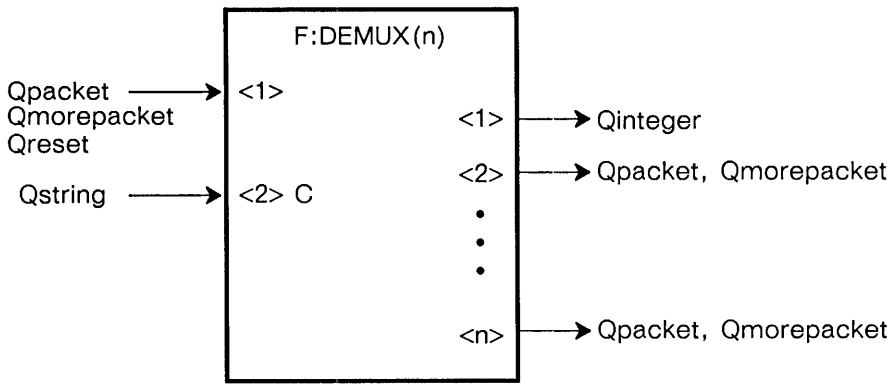
The first input <1> value is compared to 0 (zero).

NOTE

The constant delta value on input <2> may be a real number or an integer. If values on input <1> are real numbers or vectors, the delta value on input <2> must be real. If input <1> is an integer, input <2> must also be an integer.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

F:DEMUX(n) demultiplexes a stream of Qpackets/Qmorepackets from input <1> to one of the n output channels. The first byte of an incoming Qpacket is assumed to be the multiplexing byte - equal to the base character (from input <2>) + K, where K is the channel number. If K > (n–3) or K < 0, there is no channel for this output and a pair of messages are sent on outputs <1> and <2>. These can be used to allow for later remultiplexing or further demultiplexing. An integer giving the indicated output port is sent on output <1> and the message for which there was no defined output is sent on output <2>. Whether or not K is within the limits implied by the number of outputs of DEMUX, the multiplexing byte is removed from the start of the packet.

F:DEMUX(n) passes incoming Qmorepackets out the current channel (as defined by the last Qpacket). Initially, after a Qreset is received, the current channel is –1.

When instancing this function, a parameter is required to specify the number of outputs.

DESCRIPTION

INPUTS

<1> — Qpacket - switch multiplexing channel & send
Qmorepacket - send on current channel
Qreset - current channel becomes -1 (invalid)

<2> — Qstring - base character of multiplexing byte (constant)

OUTPUTS

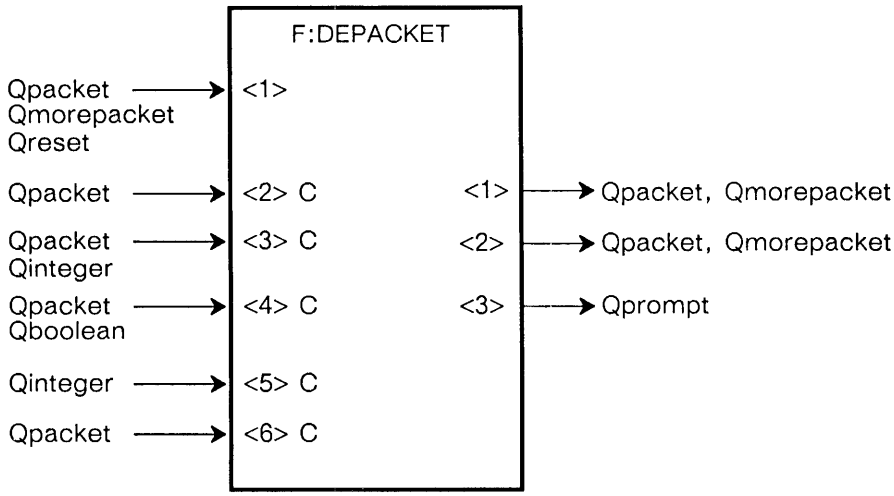
<1> — Qinteger - (i) when output port <3+i> doesn't exist

<2> — Qpacket, Qmorepacket - stream which didn't have any valid destination

<3+i> — Qpacket, Qmorepacket - output stream (i) where destination was given as [base-char + (i)]

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

F:DEPACKET converts streams of incoming bytes to Qpacket/Qmorepacket packages.

DESCRIPTION

INPUTS

<1> — Qpacket - source stream
Qmorepacket - treated as if it were Qpacket
Qreset - Return to initial state (in-between packets); send Qreset out <1>

<2> — Qpacket - FS: start packet character (constant)

<3> — Qpacket - ESC: escape character, if ESC mode (constant)
Qinteger - # count bytes, if count mode (the type of message on <3> controls ESC versus count mode) (constant)

<4> — Qpacket - base character, if count mode (constant)
Qboolean (default: FALSE) - is FS escaped?, if escape mode (constant)
FALSE: <FS> starts packet
TRUE: <ESC> <FS> starts packet

<5> — Qinteger - radix (default 10) if count mode (constant)

<6> — Qpacket - Auto-mux prefix for between-packet streams (constant)

OUTPUTS

<1> — Qpacket, Qmorepacket - packet stream

<2> — Qpacket, Qmorepacket - between-packet stream (In ESC mode, once a packet is detected, nothing will ever be between packets again.)

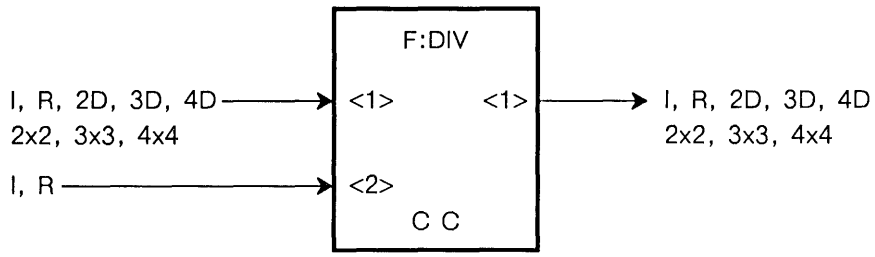
<3> — Qprompt - pass through from <1> if any show up. SR_depacket un-escapes the contents of a packet, converting <ESC> <x> to <x>

EXAMPLE

Refer to Helpful Hint 7 in Section *TT2*.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs and produces an output that is the quotient of the two inputs (input <1> is divided by input <2>).

DESCRIPTION

INPUTS

<1> — dividend

<2> — divisor

OUTPUT

<1> — quotient

NOTE

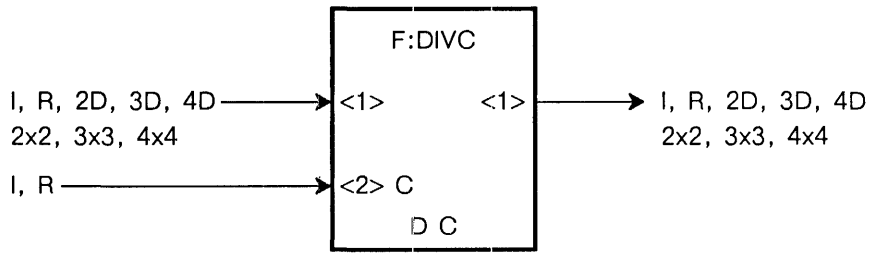
The output is the same data type as input <1> (except when input <1> is an integer and input <2> is a real number; then a real number is output). Input <2> should not be 0.

ASSOCIATED FUNCTIONS

F:DIVC, F:CDIV

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs and produces an output that is the quotient of the two inputs (input <1> is divided by input <2>). Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — dividend

<2> — divisor (constant)

OUTPUT

<1> — quotient

NOTE

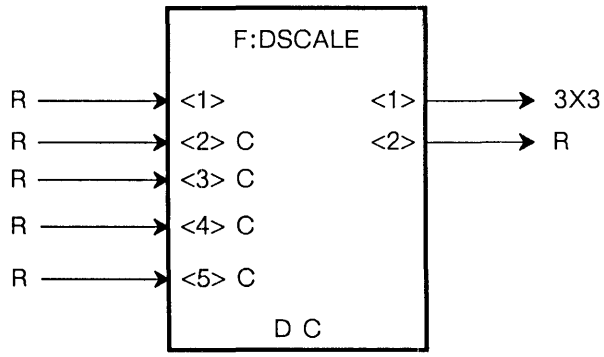
The output is the same data type as input <1> (except when input <1> is an integer and input <2> is a real number; then a real number is output). Input <2> should not be 0.

ASSOCIATED FUNCTIONS

F:DIV, F:CDIV

TYPE

Intrinsic User Function — Object Transformation



PURPOSE

Typically accepts real values originating from a control dial on input <1> and forms a 3x3 scaling matrix (output <1>) from the product of accumulated real values (input <1>) and the scaling factor on input <3>. Upper and lower scaling limits may be set on inputs <4> and <5>, respectively. If the accumulator content exceeds the upper limit (input <4>), then the upper limit value is sent out on output <1>. Likewise, if the product is below the lower limit, the lower limit value is sent out on output <1>.

DESCRIPTION

INPUTS

<1> — delta
<2> — accumulator set (constant)
<3> — scaling factor (constant)
<4> — upper limit (constant)
<5> — lower limit (constant)

OUTPUTS

<1> — 3x3 scaling matrix
<2> — accumulator contents

DEFAULTS

Inputs <3>, <4>, and <5> are optional. If input <3> receives no messages, a scaling factor of 1 is the default value. If inputs <4> and/or <5> receive no messages, no upper and/or lower limits are set.

NOTES

1. Input <2> is the accumulator. This value may be reset at any time (and is usually set initially to 1). The current accumulator content is output on output <2>.
2. It is sometimes valuable to limit the upper range of scaling to a value that will not cause data to overflow the viewport. Also, lower limits may be set to keep the object to a size that allows the object to be viewed easily and to prevent negative scaling.

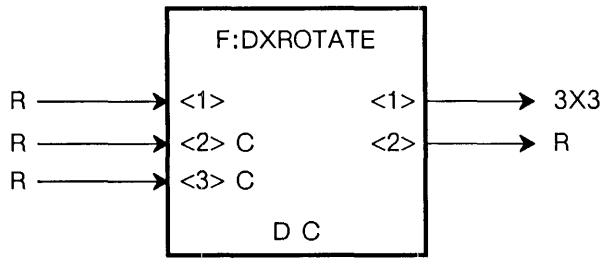
EXAMPLE

Refer to Application Note 6 in Section *TT1*.

F:DXROTATE

TYPE

Intrinsic User Function — Object Transformation



PURPOSE

Typically accepts real values originating from a control dial on input <1> and produces a 3x3 rotation matrix (output <1>) from the angle derived from the accumulated sum of the real values on input <1>, multiplied by the scale factor received on input <3>. Rotation is around the X axis.

DESCRIPTION

INPUTS

<1> — rotation delta

<2> — initial accumulator value (constant)

<3> — scale factor (constant)

OUTPUTS

<1> — 3x3 rotation matrix in X

<2> — current accumulator value

DEFAULT

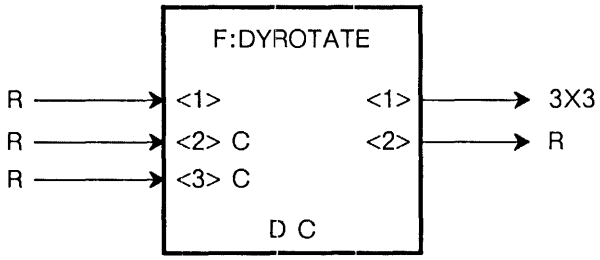
If input <3> receives no messages, a scale factor of 1 is the default value.

NOTE

Input <2> is the accumulator. This value may be reset at any time (and is usually set initially to 0). The current accumulator value is output on output <2>.

TYPE

Intrinsic User Function — Object Transformation



PURPOSE

Typically accepts real values originating from a control dial on input <1> and produces a 3x3 rotation matrix (output <1>) from the angle derived from the accumulated sum of the real values on input <1>, multiplied by the scale factor received on input <3>. Rotation is around the Y axis.

DESCRIPTION

INPUTS

<1> — rotation delta
<2> — initial accumulator value (constant)
<3> — scale factor (constant)

OUTPUTS

<1> — 3x3 rotation matrix in Y
<2> — current accumulator value

DEFAULT

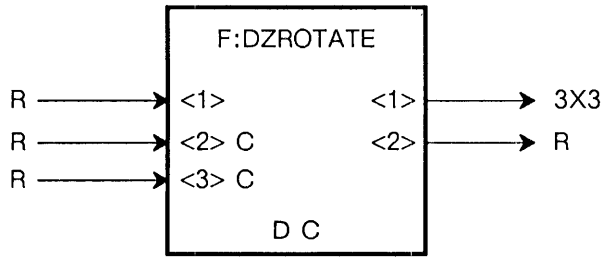
If input <3> receives no messages, a scale factor of 1 is the default value.

NOTE

Input <2> is the accumulator. This value may be reset at any time (and is usually set initially to 0). The current accumulator value is output on output <2>.

TYPE

Intrinsic User Function — Object Transformation



PURPOSE

Typically accepts real values originating from a control dial on input <1> and produces a 3x3 rotation matrix (output <1>) from the angle derived from the accumulated sum of the real values on input <1>, multiplied by the scale factor received on input <3>. Rotation is around the Z axis.

DESCRIPTION

INPUTS

<1> — rotation delta
<2> — initial accumulator value (constant)
<3> — scale factor (constant)

OUTPUTS

<1> — 3x3 rotation matrix in Z
<2> — current accumulator value

DEFAULT

If input <3> receives no messages, a scale factor of 1 is the default value.

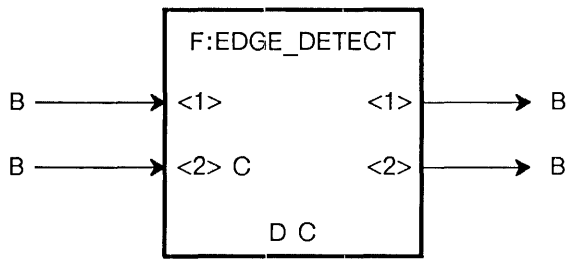
NOTE

Input <2> is the accumulator. This value may be reset at any time (and is usually set initially to 0). The current accumulator content is output on output <2>.

F:EDGE_DETECT

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

Accepts Boolean values on inputs <1> and <2>. Input <2> is a constant. Whenever the state of the Boolean value on input <1> changes to match the state on input <2>, the Boolean value on input <1> is output on <1>, and the complement of that value is output on output <2>.

DESCRIPTION

INPUTS

<1> — Boolean value

<2> — Boolean value (constant)

OUTPUTS

<1> — Boolean value on input <1> when this matches input <2>

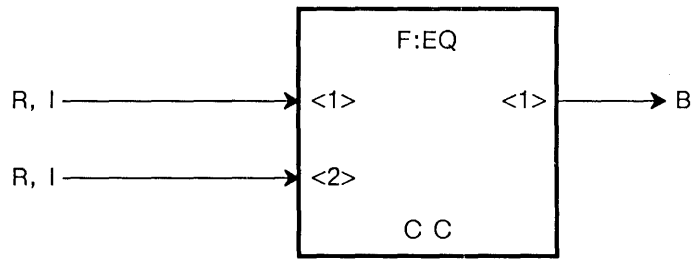
<2> — complement of output <1>

NOTE

By connecting output <2> to input <2>, all transitions are detected.

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> equals input <2>, and FALSE otherwise.

DESCRIPTION

INPUTS

<1> — real number or integer to be compared

<2> — real number or integer to be compared

OUTPUT

<1> — TRUE if input <1> equals input <2>, else FALSE

NOTE

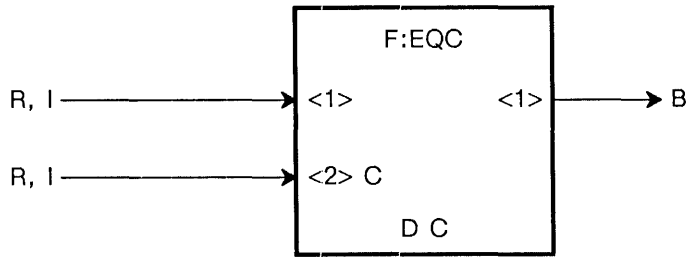
Inputs do not have to be of the same data type.

ASSOCIATED FUNCTION

F:EQC

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs, and produces a Boolean value output that is TRUE if input <1> equals input <2>, and FALSE otherwise. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — real number or integer to be compared

<2> — real number or integer to be compared (constant)

OUTPUT

<1> — TRUE if input <1> equals input <2>, else FALSE

NOTE

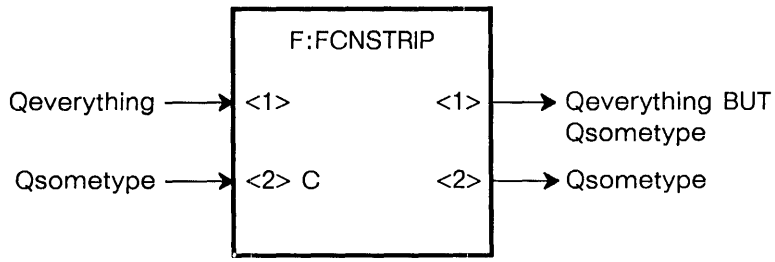
Inputs do not have to be of the same data type.

ASSOCIATED FUNCTION

F:EQ

TYPE

Intrinsic User Function — Data Selection and Manipulation

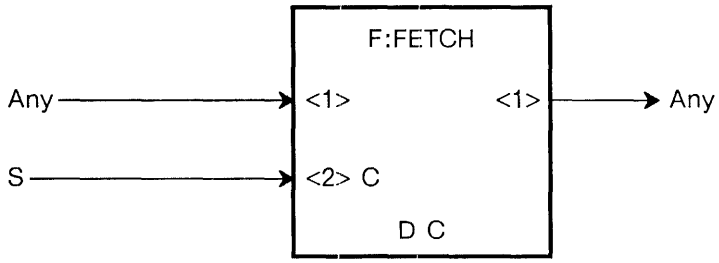


PURPOSE

F:FCNSTRIP is used to either filter out some Qdata type or, alternately, to select a given Qdata type. The type of incoming messages on <1> is compared to the type of the message on the constant queue <2>. If the types are different, then the incoming message is sent out output <1>, thus filtering out the type on input <2>. If they are of the same type, then output is to queue <2>, effectively selecting that type.

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

Accepts a string which is the name of a variable on input <2>. When any message is received on input <1>, the message currently stored in the variable named on input <2> is fetched and output from this function. The message stored in the named variable may be of any data type. The arrival of input <1> is used to fire the function, but is otherwise ignored. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — trigger

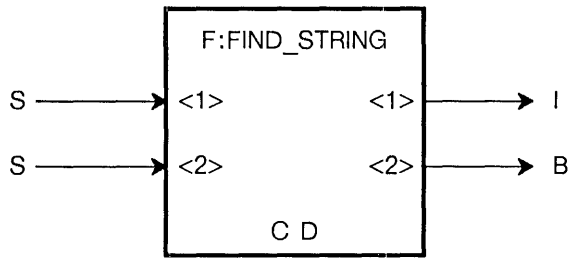
<2> — variable name (constant)

OUTPUT

<1> — message associated with variable name on input <2>

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

If the string on input <2> is a substring of the string on input <1>, the starting position of the substring and a Boolean TRUE are output. A FALSE is output if the substring cannot be found and nothing is sent on output <1>.

DESCRIPTION

INPUTS

<1> — string

<2> — substring

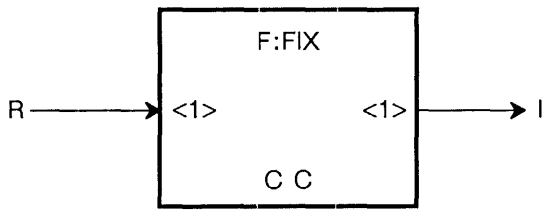
OUTPUTS

<1> — starting position of the substring, if found

<2> — TRUE = substring found, FALSE = not found

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts a real number and outputs a value that is truncated to an integer (toward zero).

DESCRIPTION

INPUT

<1> — real number

OUTPUT

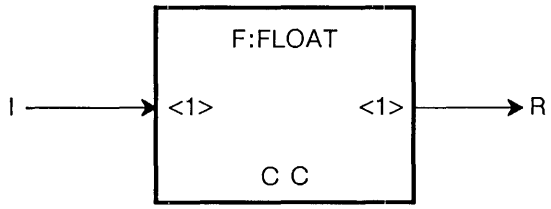
<1> — real on input <1> truncated to an integer

ASSOCIATED FUNCTION

F:CEILING

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts an integer and outputs a real number of the same value.

DESCRIPTION

INPUT

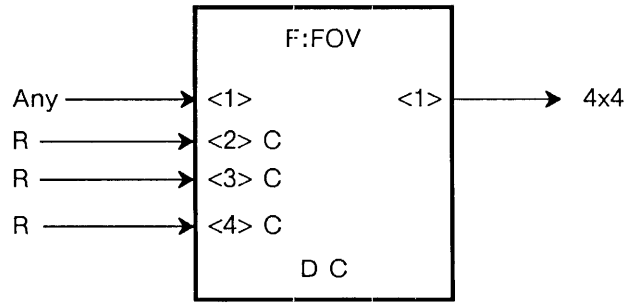
<1> — integer

OUTPUT

<1> — real number of the same value as input <1>

TYPE

Intrinsic User Function — Viewing Transformation



PURPOSE

This is the functional counterpart of the FIELD_OF_VIEW command. The field of view that is specified by this function is used for perspective projections.

DESCRIPTION

INPUTS

<1> — trigger

<2> — viewing angle (constant)

<3> — front boundary (constant)

<4> — back boundary (constant)

OUTPUT

<1> — 4x4 matrix

NOTES

1. The message on input <1> acts as a trigger to the function.
2. The constant real value on input <2> represents the viewing angle in degrees. This angle defines the viewing frustum.

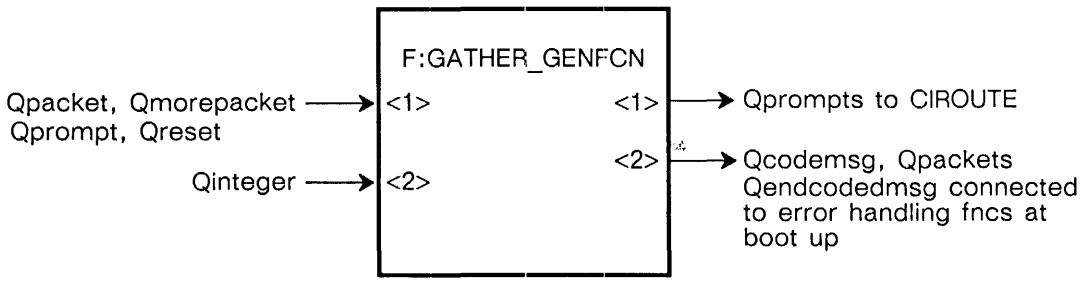
3. The front boundary and back boundary of the viewing frustum are specified as constant real numbers on inputs <3> and <4>, respectively.
4. The field of view specified on the inputs to F:FOV is output as a 4x4 matrix.

ASSOCIATED FUNCTIONS

F:WINDOW, F:MATRIX4

TYPE

Intrinsic User Function — Miscellaneous



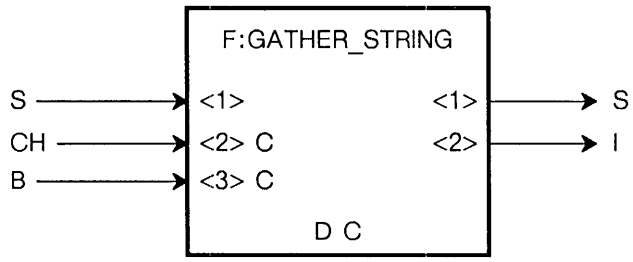
PURPOSE

F:GATHER_GENFCN is used to download the code for user-written functions. The first messages contain information about the user-written function, and the remainder contain Motorola S-records. It gathers the data specifying a user-written function on input <1> and creates a new function.

Input <2> receives a CI number to associate the function with an instance of the CI. When the CI receives an INIT command, it will remove the functions created by its associated gather_genfunction. This number corresponds to the parameter given in instancing the F:CI function. Numbers less than 10 are reserved for system use. The number 4 is the default.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Collects strings that arrive at input <1> until the terminator character on input <2> arrives. Concatenates all strings into one packet and outputs the concatenated string on output <1>. If the Boolean value on input <3> is TRUE, the terminator character is appended to the string. Output <2> contains the length of the string. Inputs <2> and <3> are constants.

DESCRIPTION

INPUTS

<1> — string

<2> — packet terminator (constant)

<3> — TRUE = with terminator, FALSE = without terminator (constant)

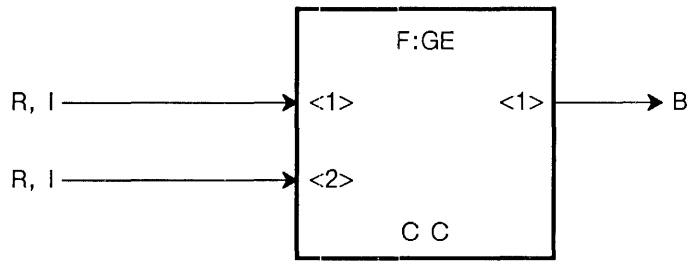
OUTPUTS

<1> — concatenated string (packet)

<2> — length of the string

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> is greater than or equal to input <2> and FALSE otherwise.

DESCRIPTION

INPUTS

<1> — value to be compared

<2> — value to be compared

OUTPUT

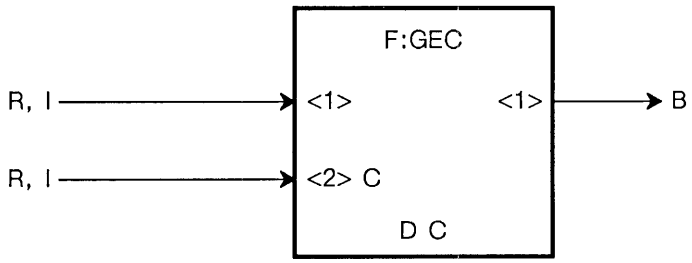
<1> — TRUE if input <1> is greater than or equal to input <2>, otherwise FALSE

ASSOCIATED FUNCTIONS

F:GEC, F:CGE

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> is greater than or equal to input <2> and FALSE otherwise. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — value to be compared

<2> — value to be compared (constant)

OUTPUT

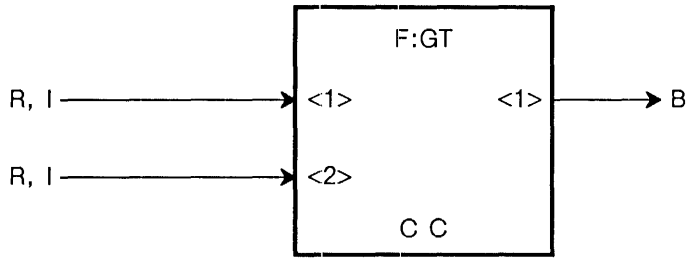
<1> — TRUE if input <1> is greater than or equal to input <2>, otherwise FALSE

ASSOCIATED FUNCTIONS

F:GE, F:CGE

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> is greater than input <2> and FALSE otherwise.

DESCRIPTION

INPUTS

<1> — value to be compared

<2> — value to be compared

OUTPUT

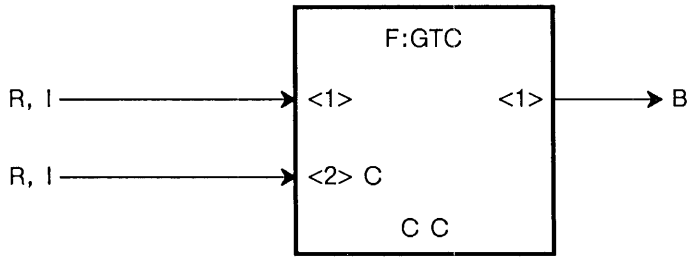
<1> — TRUE if input <1> greater than input <2>, otherwise FALSE

ASSOCIATED FUNCTIONS

F:GTC, F:CGT

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> is greater than input <2> and FALSE otherwise. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — value to be compared

<2> — value to be compared (constant)

OUTPUT

<1> — TRUE if input <1> greater than input <2>, otherwise FALSE

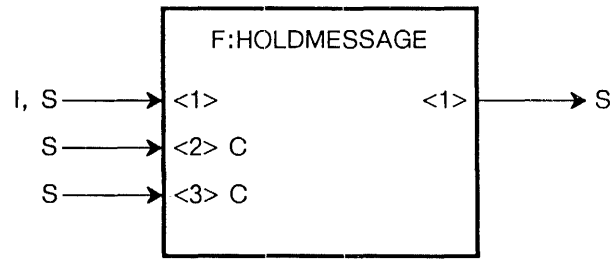
ASSOCIATED FUNCTIONS

F:GT, F:CGT

F:HOLDMESSAGE

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

This function is used to send all messages from the PS 390 to the host when using the GSRs.

DESCRIPTION

INPUTS

<1> — integer or any string
<2> — any string (constant)
<3> — any string (constant)

OUTPUT

<1> — any string

DEFAULTS

The default input value for inputs <2> and <3> is a carriage return.

NOTES

1. Input <1> contains the Qpackets of messages to be sent to the host and Qintegers used to trigger the messages as follows:

   Fix(0) clears any messages waiting in the queue of messages to be sent to the host.

   Fix(1) sends a message if it is waiting. Otherwise, the message "no-messages" is sent as determined by input <3>.

   Fix(2) sends a message if it is waiting. Otherwise, it waits until a Qpacket message arrives on input <1> and sends the message immediately.

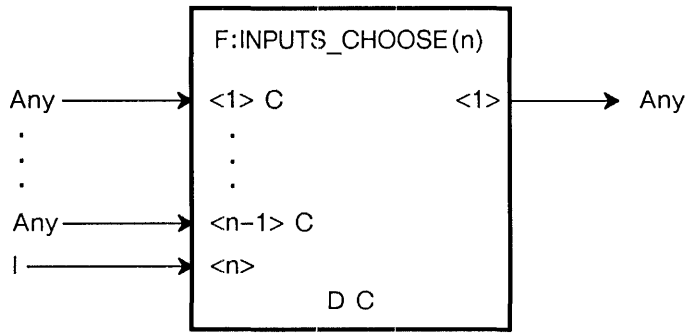
2. Input <2> contains the message terminator Qpacket that is added to the end of messages arriving on input <1> just prior to transmission to the host.

3. Input <3> contains the "no-messages" Qpacket. If the function receives a Fix(1) on input <1>, then the message on this constant queue is sent only if there are no other messages waiting to be sent on input <1>. Otherwise, the first message on the queue of messages is sent from output <1> with the message terminator Qpacket as defined on input <2>.

4. Output <1> sends the message to the host in response to the receipt of either a Fix(1) or Fix(2) on input <1>.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts an integer with a value from 1 to (n–1) on input <n> and uses that value to choose which of inputs <1> through <n–1> to accept as an input. The chosen message is then output.

DESCRIPTION

INPUTS

<1> — any message (constant)

•

•

•

<n–1>— any message (constant)

<n> — chosen message number

OUTPUT

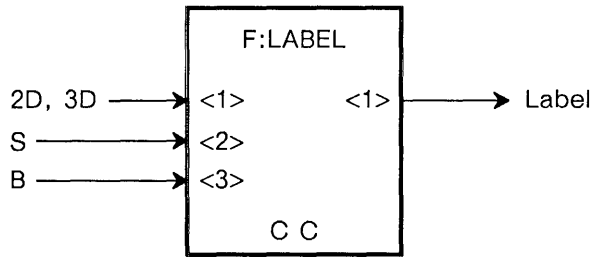
<1> — chosen message

NOTE

To set up F:INPUTS_CHOOSE(n) for a given number of messages between 2 and 127 inclusive, add one to the number of messages and substitute the result for "n" in the function identifier. For example, F:INPUTS_CHOOSE(5) accepts four messages at inputs <1> through <4>. The selector input is always input <n>. Thus, for F:INPUTS_CHOOSE(5), the selector input is <5>.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Creates a label to send to a LABELS node using the vector on input <1> as the position of the label and the string on input <2> as the text of the label. Input <3> indicates whether the label is displayed or not.

DESCRIPTION

INPUTS

<1> — X, Y, and (optionally) Z location of the label

<2> — text of the label

<3> — TRUE = displayed, FALSE = not displayed

OUTPUT

<1> — label for input to a LABELS node

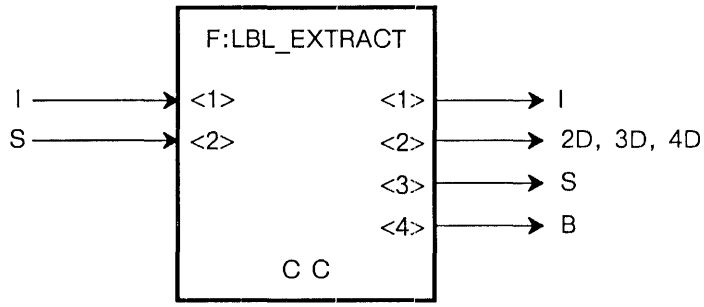
NOTE

The data type output by this function can only be used to update a labels node. It is not accessible or printable.

F:LBL_EXTRACT

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Extracts information about a string from a LABELS node given an index into the labels block on input <1> and the name of the labels node on input <2>.

DESCRIPTION

INPUTS

<1> — index of the string in question

<2> — name of the LABELS node

OUTPUTS

<1> — data type

<2> — the start location of the string in question

<3> — the text of the string

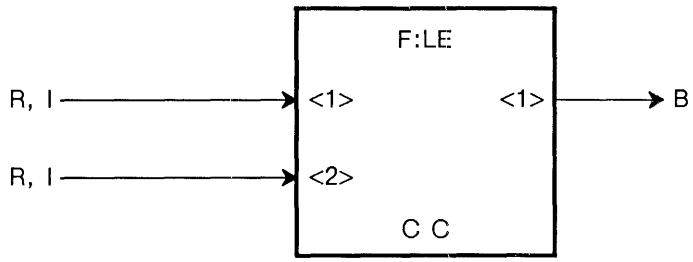
<4> — TRUE = on, FALSE = off

NOTES

1. The integer on output <1> is the same as would be sent from output <7> of F:PICKINFO.
2. Output <4> indicates whether the string is on or off.

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> is less than or equal to input <2> and FALSE otherwise.

DESCRIPTION

INPUTS

<1> — value to be compared

<2> — value to be compared

OUTPUT

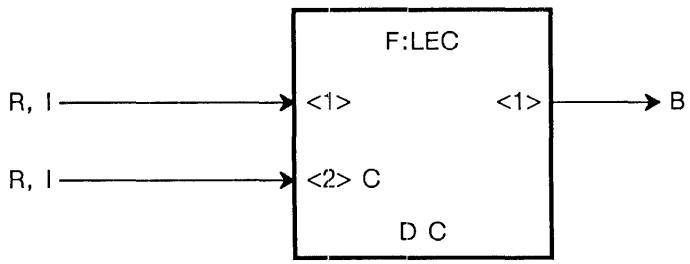
<1> — TRUE if input <1> is less than or equal to input <2>, otherwise FALSE

ASSOCIATED FUNCTIONS

F:LEC, F:CLE

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> is less than or equal to input <2> and FALSE otherwise. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — value to be compared

<2> — value to be compared (constant)

OUTPUT

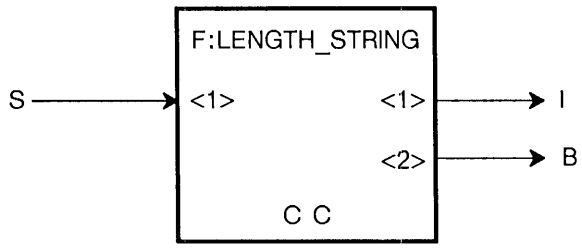
<1> — TRUE if input <1> is less than or equal to input <2>, otherwise FALSE

ASSOCIATED FUNCTIONS

F:LE, F:CLE

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Outputs the length of a string.

DESCRIPTION

INPUT

<1> — string

OUTPUTS

<1> — length of the string

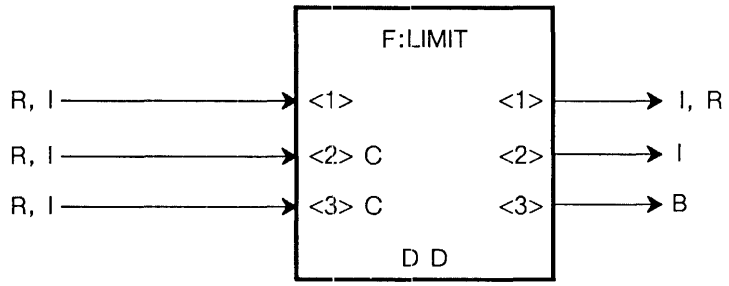
<2> — TRUE = null string, FALSE otherwise

NOTE

A possible output is zero.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts real number or integer values on all inputs; all three input values must be of the same data type. The output data type is the same as the input data type.

DESCRIPTION

INPUTS

<1> — value

<2> — upper limit (constant)

<3> — lower limit (constant)

OUTPUTS

<1> — input <1> if this value is in range

<2> — in-range value

<3> — TRUE if in-range, FALSE if out-of-range

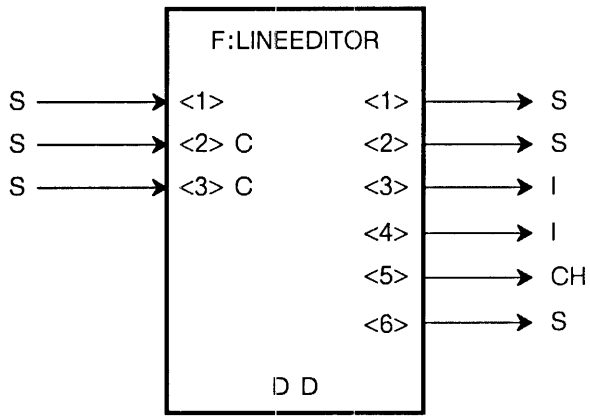
NOTES

1. The value on input <1> is compared to the constant upper limit value on input <2> and the constant lower limit value on input <3>.
2. If the input <1> value is in range, that value is output unchanged on output <1> and output <2>, and a TRUE is output on <3>.

3. If the input <1> value is out of range, the output <1> value is adjusted to the nearer limit (as set by inputs <2> and <3>), output <2> is disabled, and output <3> is FALSE.
4. If the value on input <2> is less than or equal to the value on input <3>, the function will always output the value received on input <3>.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts a stream of characters and simple editing commands, accumulates the characters in an internal line buffer, applies the commands to the contents of the line buffer as they are received, and outputs the edited line when a specified delimiter character is recognized.

DESCRIPTION

INPUTS

<1> — editing commands and material to be edited (input string)
<2> — prompt message (constant)
<3> — line delimiter (constant)

OUTPUTS

<1> — edited output
<2> — display output
<3> — integer for <clear> of CHARACTERS
<4> — integer for <delete> of CHARACTERS
<5> — character for <append> of CHARACTERS
<6> — string for <substitute> or <replace> of CHARACTERS

NOTES

1. In a typical application, F:LINEEDITOR receives its input from the PS 390 keyboard and sends its edited output either to a terminal (such as the debug terminal or the terminal emulator) or to a CHARACTERS node in the PS 390 display structure. A specially formatted "display" output is used for terminals; other outputs are intended as connections into CHARACTERS to allow keyboard editing of a CHARACTERS string.

2. F:LINEEDITOR recognizes the following editing commands:

   Delete (Hex '7F'): Deletes the most recently received character from the internal line buffer.

   CTRL/U (Hex '15'): Deletes the entire line buffer. Redisplays a predetermined prompt message at any associated terminals by sending the prompt string on the display output <2>.

   CTRL/R (Hex '12'): Retypes the entire line (preceded by the prompt message) at any associated terminals by sending the prompt and line along the display output <2>.

3. Input <1> receives the stream of strings to be collected and edited, along with all editing commands. The PS 390 keyboard is typically connected to this input.

4. Input <2> contains a prompt message, if one is needed. The prompt string may contain one or several characters. This prompt appears only at output <2>, and it appears there whenever a CTRL/U, a CTRL/R, or a delimiter is received at input <1>. The prompt message is optional and there is no default.

5. Input <3> contains a single character designated as the delimiter. When this character is received at input <1>, the contents of the line buffer appear at outputs <1> and <6> (edited by the editing commands), and at output <2> (along with the prompt).

6. The default delimiter is <CR> (carriage return; Hex '0D'), but this <CR> is always expanded to <CR><LF> (carriage return/line-feed; Hex '0D0A') for output at <1>, <2>, and <6>.

7. If input <3> contains a non-<CR> delimiter <delim>, this delimiter is passed on as is to outputs <1> and <6>, but it is always converted to <delim><CR><LF> for output <2> (the display output). (This implies that specifying a delimiter of <LF> produces double-spaced display output.)

8. Output <1> contains the contents of the line buffer, which in turn is composed of the collected and edited characters from input <1>. This output fires when a delimiter is recognized at input <1> or when 255 characters have been collected since the last firing or since initialization.

9. Output <2> is the display output. Unlike outputs <1> and <6>, this output includes "editing effects" intended for terminal display (prompt messages, displayed CTRL/Us and CTRL/Rs, character erasures corresponding to deletes, and so on). For the treatment of delimiters at output <2>, see note 7 above.

10. Output <3> is an integer output intended as a connection into the <clear> input of a CHARACTERS command. The integer is sent whenever a CTRL/U is received at input <1>.

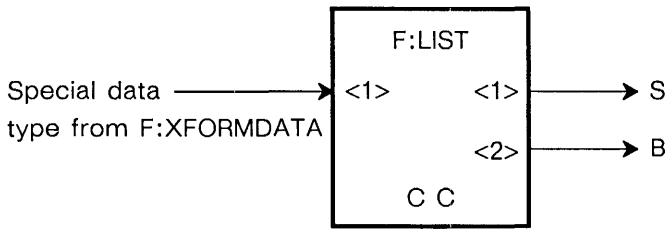
11. Output <4> always sends an integer 1 and is intended as a connection into the <delete> input of a CHARACTERS command. The 1 is sent whenever a delete is received at input <1>.

12. Output <5> is intended as a connection into the <append> input of a CHARACTERS command. This output passes on all characters received at input <1> except editing commands (delete, CTRL/U, CTRL/R). No buffering is performed at this output—it fires once for each non-command character, and the message is always a single character.

13. Output <6> is intended as a connection into the <substitute> or <replace> input of a CHARACTERS command. It fires whenever the function is activated by a (single-character or multi-character) string at input <1>. In addition, output <6> fires whenever output <1> fires.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Converts the output of the F:XFORMDATA function to an ASCII string. This function is always used with F:XFORMDATA.

DESCRIPTION

INPUT

<1> — data output by F:XFORMDATA

OUTPUTS

<1> — resulting ASCII string

<2> — Boolean value (TRUE)

NOTES

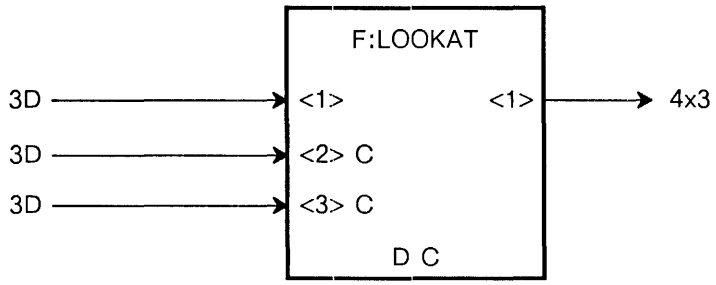
1. Input <1> is always connected to output <1> of F:XFORMDATA.
2. Output <2> is TRUE when processing is complete. There is no output otherwise.
3. Output <2> should be connected to an instance of F:SYNC(2) to synchronize F:LIST completion with the initiation of a subsequent transformed-data request.

EXAMPLE

Refer to Helpful Hint 4 in Section *TT2*.

TYPE

Intrinsic User Function — Viewing Transformation



PURPOSE

Accepts three 3D vectors that specify the position to "look at," the position to "look from," and which direction is "up." Inputs <2> and <3> ("look from" and "up" orientation) are constants.

DESCRIPTION

INPUTS

<1> — look at point

<2> — look from point (constant)

<3> — up orientation (constant)

OUTPUT

<1> — 4x3 viewing matrix

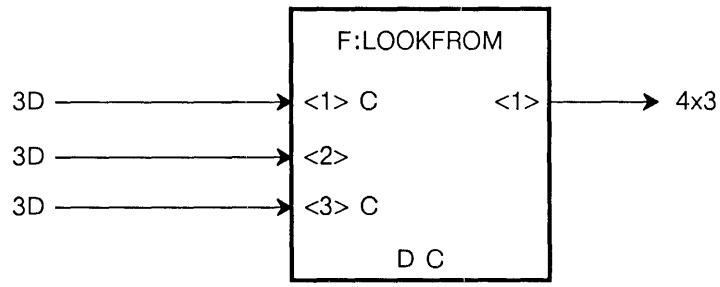
NOTES

1. Input <1>, the "look at" vector, triggers the function.
2. The 3D vectors are used to generate a 4x3 matrix that may be used to update a LOOK viewing-transformation node in a display structure.

F:LOOKFROM

TYPE

Intrinsic User Function — Viewing Transformation



PURPOSE

Accepts three 3D vectors that specify the position to "look at," the position to "look from," and which direction is "up." Inputs <1> and <3> ("look at" and "up" orientation) are constants.

DESCRIPTION

INPUTS

<1> — look at point (constant)

<2> — look from point

<3> — up orientation (constant)

OUTPUT

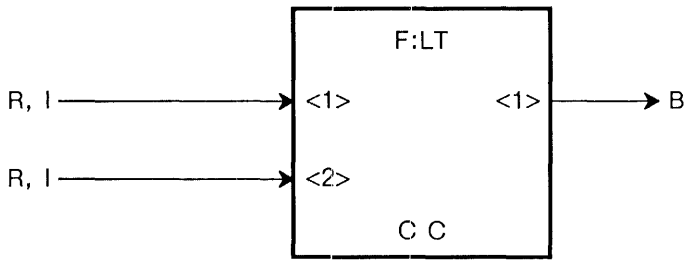
<1> — 4x3 viewing matrix

NOTES

1. Input <2>, the "look from" vector, triggers the function.
2. The 3D vectors are used to generate a 4x3 matrix that may be used to update a LOOK viewing-transformation node in a display structure.

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> is less than input <2> and FALSE otherwise.

DESCRIPTION

INPUTS

<1> — value to be compared

<2> — value to be compared

OUTPUT

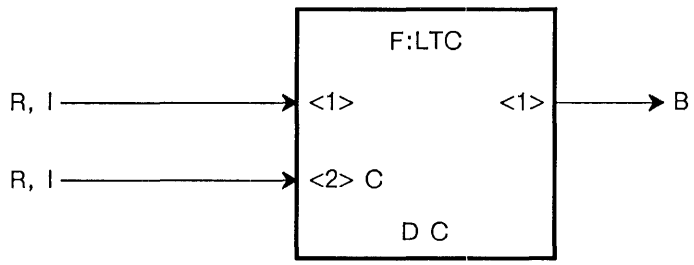
<1> — TRUE if input <1> is less than input <2>, otherwise FALSE

ASSOCIATED FUNCTIONS

F:LTC, F:CLT

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> is less than input <2> and FALSE otherwise. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — value to be compared

<2> — value to be compared (constant)

OUTPUT

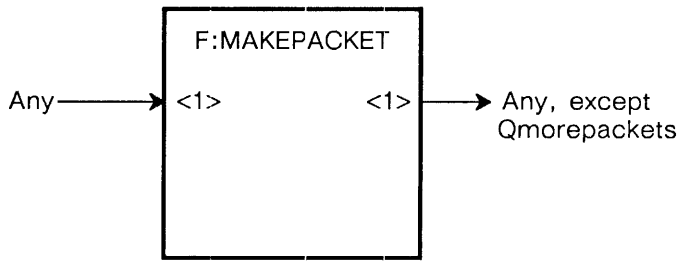
<1> — TRUE if input <1> is less than input <2>, otherwise FALSE

ASSOCIATED FUNCTIONS

F:LT, F:CLT

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

This function is used to convert a Qmorepacket received on input <1> to a Qpacket on output <1>. All other messages are passed through the function unchanged.

DESCRIPTION

INPUT

<1> — any message

OUTPUT

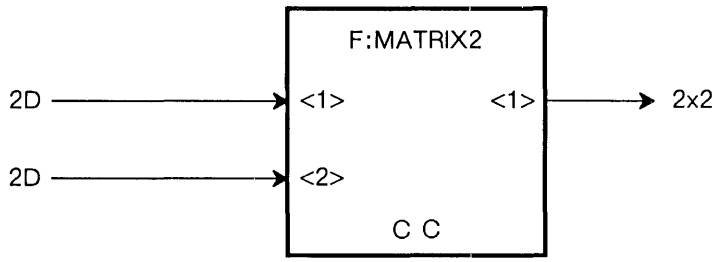
<1> — any message except Qmorepacket

NOTE

F:CIROUTE(n) outputs both Qpackets and Qmorepackets. Since some functions can accept only Qpackets, F:MAKEPACKET can be used to accept output from F:CIROUTE(n).

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts two 2D vectors and produces a 2x2 matrix.

DESCRIPTION

INPUTS

<1> — 2D vector

<2> — 2D vector

OUTPUT

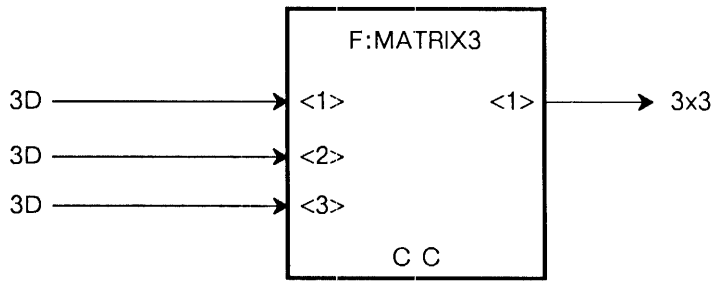
<1> — 2x2 matrix

NOTES

1. The matrix output may be used to update a 2x2 matrix node in a display structure or as input to another function.
2. The vector on input <1> is output as the first row of the matrix. The vector on input <2> is output as the second row.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts three 3D vectors and produces a 3x3 matrix.

DESCRIPTION

INPUTS

<1> — 3D vector

<2> — 3D vector

<3> — 3D vector

OUTPUT

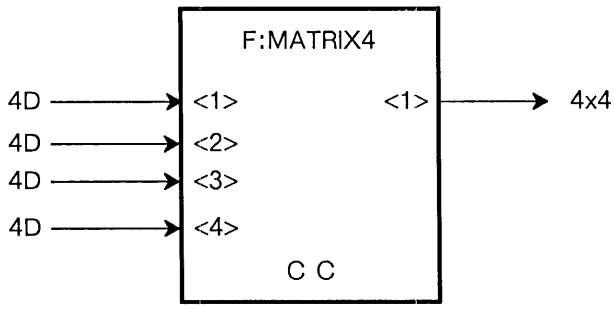
<1> — 3x3 matrix

NOTES

1. The matrix output may be used to update a 3x3 matrix node in a display structure or as input to another function.
2. The vector on input <1> is output as the first row of the matrix. The vector on input <2> is output as the second row. The vector on input <3> is the third row.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts four 4D vectors and produces a 4x4 matrix.

DESCRIPTION

INPUTS

<1> — 4D vector

<2> — 4D vector

<3> — 4D vector

<4> — 4D vector

OUTPUT

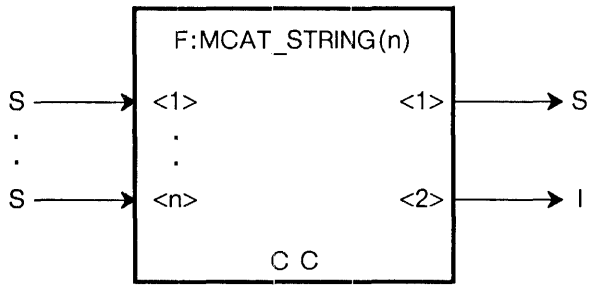
<1> — 4x4 matrix

NOTES

1. The matrix output may be used to update a 4x4 matrix node in a display structure or as input to another function.
2. The vector on input <1> is output as the first row of the matrix. The vector on input <2> is output as the second row. The vector on input <3> is the third row. The vector on input <4> is the fourth row.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts strings on inputs <1> through <n> and concatenates them into a single string. Output <1> contains the resulting string and output <2> contains its length.

DESCRIPTION

INPUTS

<1> — string

•

•

•

<n> — string

OUTPUTS

<1> — concatenated string

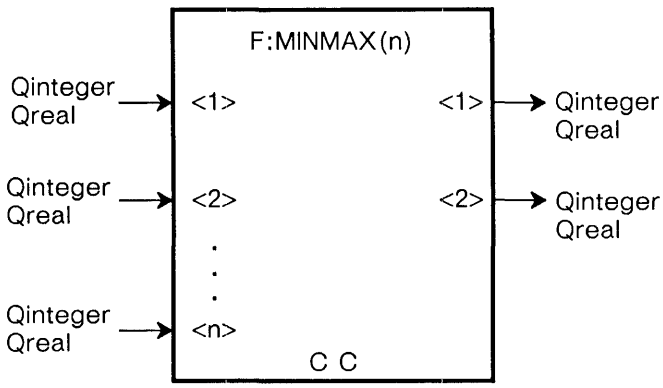
<2> — string length

NOTE

The limit to the number of inputs to this function is 127.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Selects the minimum and maximum values on inputs. The number of inputs is indicated by the parameter (n).

DESCRIPTION

INPUTS

<1> — Qinteger/Qreal
<2> — Qinteger/Qreal
•
•
•
<n> — Qinteger/Qreal

OUTPUTS

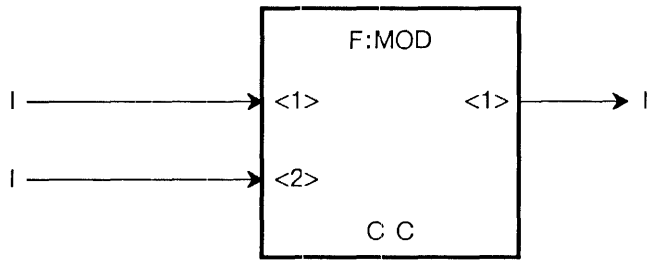
<1> — Qinteger/Qreal (maximum)
<2> — Qinteger/Qreal (minimum)

NOTE

The type of input on inputs <2> through <n> must be the same type as on input <1>.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two integers as inputs and produces an integer output that is the remainder resulting from the division of the value on input <1> by the value on input <2>. The integer on input <2> must be positive.

DESCRIPTION

INPUTS

<1> — integer

<2> — integer

OUTPUT

<1> — remainder from dividing input <1> by input <2>

NOTE

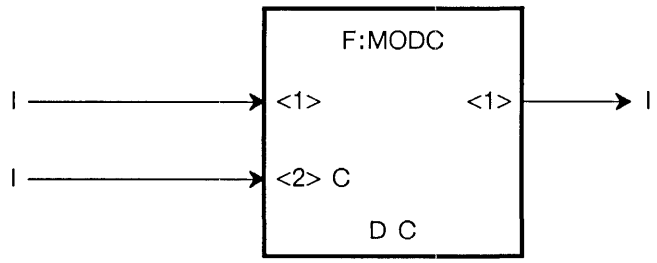
F:MOD uses a Pascal-like definition of modulo. For a negative integer on input <1>, the resulting output will be negative. For example, -8 mod 3 is -2.

ASSOCIATED FUNCTION

F:MODC

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two integers as inputs and produces an integer output that is the remainder resulting from the division of the value on input <1> by the value on input <2>. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — integer

<2> — integer (constant)

OUTPUT

<1> — remainder from dividing input <1> by input <2>

NOTE

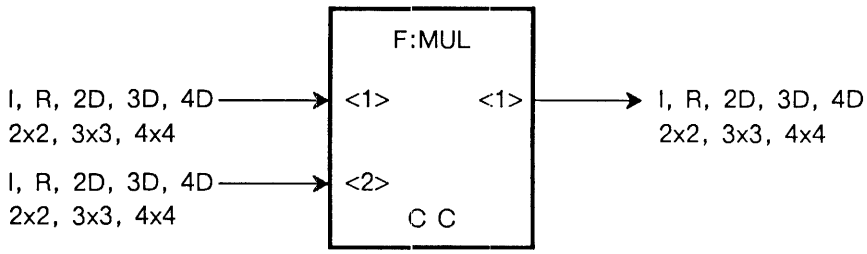
F:MODC uses a Pascal-like definition of modulo. For a negative integer on input <1>, the resulting output will be negative. For example, -8 mod 3 is -2.

ASSOCIATED FUNCTION

F:MOD

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs and outputs their product.

DESCRIPTION

INPUTS

<1> — multiplier

<2> — multiplicand

OUTPUT

<1> — product

NOTE

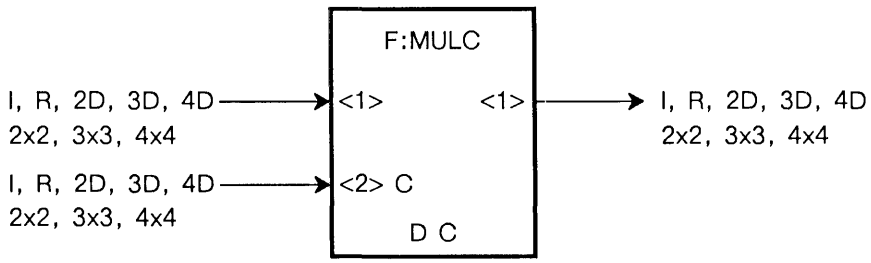
The two input values must be compatible data types; the output data type depends on the combination of input data types. Vectors are taken to be either row or column vectors, as appropriate, to perform the multiplication.

ASSOCIATED FUNCTIONS

F:MULC, F:CMUL

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs and outputs their product. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — multiplier

<2> — multiplicand (constant)

OUTPUT

<1> — product

NOTE

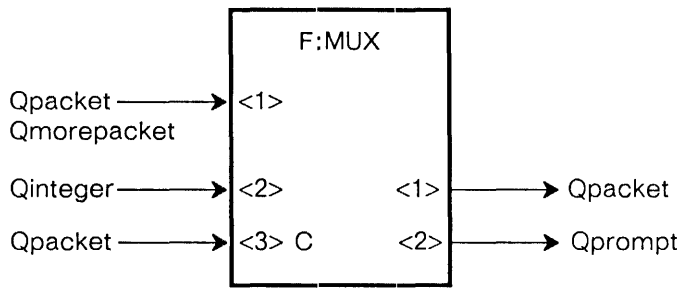
The two input values must be compatible data types; the output data type depends on the combination of input data types. Vectors are taken to be either row or column vectors, as appropriate, to perform the multiplication.

ASSOCIATED FUNCTIONS

F:MUL, F:CMUL

TYPE

Intrinsic User Function — Data Selection and Manipulation

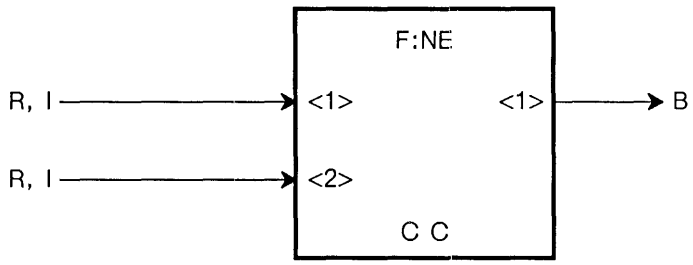


PURPOSE

F:MUX is the inverse of F:DEMUX. It accepts Qpacket/Qmorepackets and prefixes each incoming bundle of types with the multiplexing character, the base character (from input <3>) plus the channel number from input <2>. The Qprompt on output <2> is any Qprompt which showed up on input <1>.

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> is not equal to input <2> and FALSE otherwise.

DESCRIPTION

INPUTS

<1> — value to be compared

<2> — value to be compared

OUTPUT

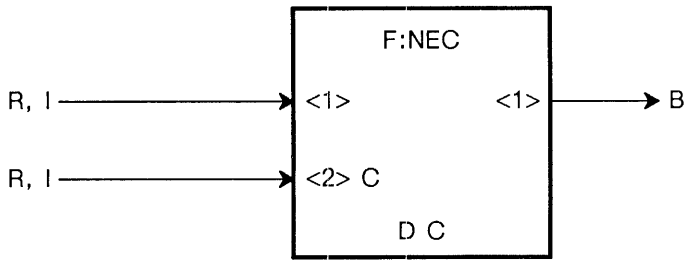
<1> — TRUE if input <1> is not equal to input <2>, otherwise FALSE

ASSOCIATED FUNCTION

F:NEC

TYPE

Intrinsic User Function — Comparison



PURPOSE

Accepts any combination of real numbers and integers on its two inputs and produces a Boolean value output that is TRUE if input <1> is not equal to input <2> and FALSE otherwise. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — value to be compared

<2> — value to be compared (constant)

OUTPUT

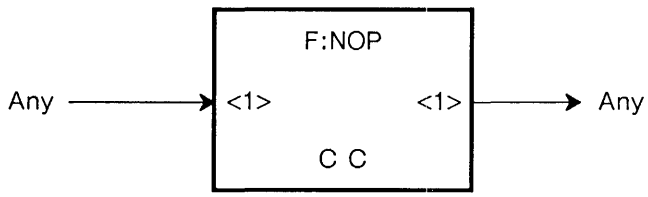
<1> — TRUE is input <1> is not equal to input <2>, otherwise FALSE

ASSOCIATED FUNCTION

F:NE

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

Accepts any message and outputs that message unchanged.

DESCRIPTION

INPUT

<1> — any message

OUTPUT

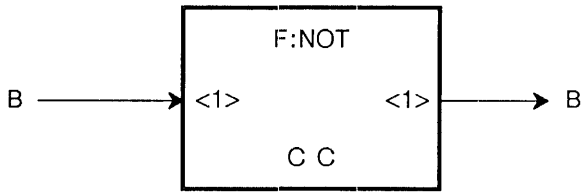
<1> — message on input <1>

NOTE

This function is useful for tying a set of many outputs to a set of many inputs.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts a Boolean value input and outputs its complement as a Boolean value.

DESCRIPTION

INPUT

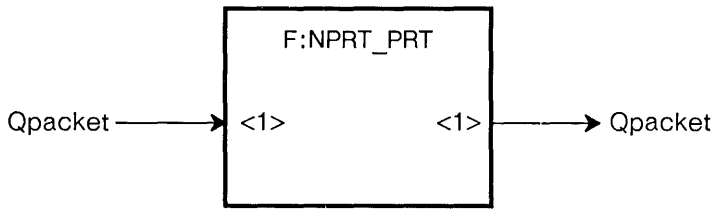
<1> — Boolean value

OUTPUT

<1> — logical complement of input <1>

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

This function converts strings containing non-printable characters to strings of printable characters. Example: ↑L to <FF>.

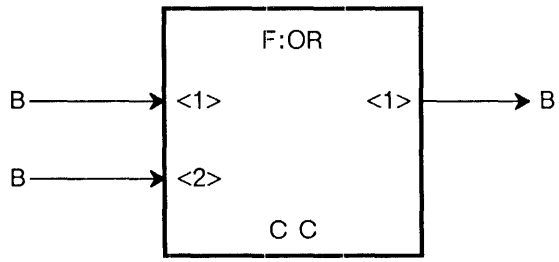
This function is a helpful debugging aid, as it allows non-printable characters to be printed. For example, this function's input could be connected to the function receiving input from the host, and its output connected to the Terminal Emulator. Then all characters that enter the PS 390 from the host could be seen.

EXAMPLE

Refer to Helpful Hint 14 in Section *TT2*.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two Boolean values as input and produces a Boolean value output that is the logical OR of the two inputs.

DESCRIPTION

INPUTS

<1> — Boolean value

<2> — Boolean value

OUTPUT

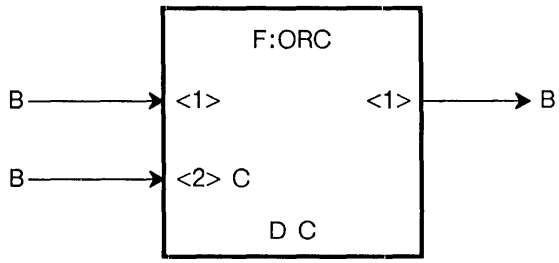
<1> — logical OR of the two inputs

ASSOCIATED FUNCTION

F:ORC

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two Boolean values as input and produces a Boolean value output that is the logical OR of the two inputs. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — Boolean value

<2> — Boolean value (constant)

OUTPUT

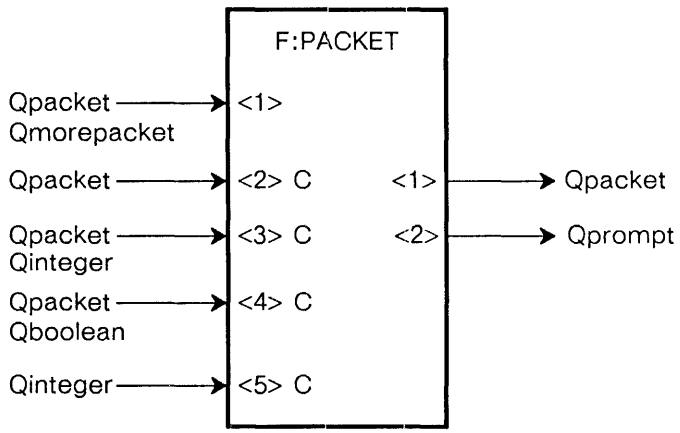
<1> — logical OR of the two inputs

ASSOCIATED FUNCTION

F:OR

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

The F:PACKET function takes incoming Qpacket messages from input <1> and prefixes each one with the proper packet header before sending them on. (It is the inverse of the F:DEPACKET function.) It routes any Qprompts arriving at input <1> off to output <2>. Like F:DEPACKET, this function can operate in either count mode or escape mode. In count mode, F:PACKET makes a packet of the following form:

FS	count bytes	message body from input <1>

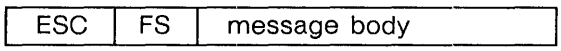
The definition of the FS character is taken from the single-character string on input <2>. The count is defined to have n bytes (n taken from input <3>). Each count type is offset from the base character (on input <4>). The radix of the count is on input <5>.

In escape mode, the packet is defined as:

FS	message body

if input <4> is false.

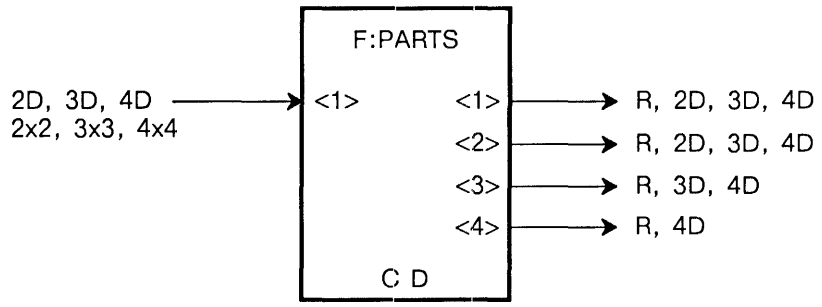
If input <4> is true, it is defined as:



The definition of FS is taken from the single-character string on input <2>. If input <4> is false, any FS character within the packet is prefixed with ESC. In either mode, any ESC byte within the packet is also prefixed with ESC.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Separates a vector into its elements or a square matrix into its row vectors.

DESCRIPTION

INPUT

<1> — any vector or matrix

OUTPUTS

<1> — X component or row vector

<2> — Y component or row vector

<3> — Z component or row vector

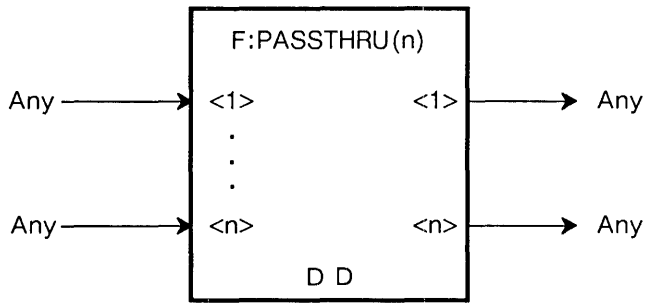
<4> — W component or row vector

NOTES

1. If a square matrix is sent to input <1>, its row vectors appear in sequence at the outputs.
2. If a vector is input, its components are output as real numbers. The X component is output on output <1>, the Y component on output <2>, and the Z and W components (if any) on output <3> and output <4> respectively.
3. Note that some outputs are not always used. For example, if a 3x3 matrix or a 3D vector is sent to F:PARTS, nothing is output on output <4>.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Immediately passes the message which arrives at any input to all function queues connected to its associated output.

DESCRIPTION

INPUTS

<1> — any message

•

•

•

<n> — any message

OUTPUTS

<1> — message on input <1>

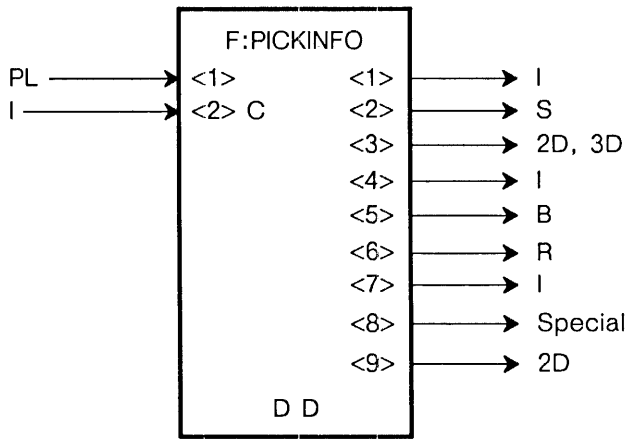
<n> — message on input <n>

NOTES

1. A message is passed through as soon as it arrives at an input queue. The function does not have to wait for messages on all its inputs before it becomes active.
2. The SETUP CNESS command cannot be used with this function.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Reformats picklist information for use by other functions. The output picklist is separated into its component parts.

DESCRIPTION

INPUTS

<1> — picklist

<2> — depth within structure reported (constant)

OUTPUTS

<1> — index

<2> — pick identifier(s)

<3> — coordinates

<4> — dimension

<5> — coordinates reported

<6> — curve parameter, t

<7> — data type code
<8> — name of picked element
<9> — screen coordinates of the picked point

DEFAULT

The default depth on input <2> is all.

NOTES

1. Input <1> accepts a picklist. Since the only source of a picklist is the initial function instance PICK, instances of F:PICKINFO must be connected to PICK.

2. Input <2> accepts an integer that specifies the depth within a structure that will be reported when a pick occurs. For example, if the picked item were at the fiftieth level within pick identifiers (i.e., the picked data could be appended with 49 pick identifiers separated by commas) and the integer 2 were input on input <2>, then only the identifier of the picked item and the item directly above it in the structure would be output as the string on output <2>.

3. The output information varies with the type of picklist supplied. If the associated PICK function instance has a TRUE on input <2>, it supplies a detailed coordinate picklist, and most or all of F:PICKINFO's outputs are activated. If the associated PICK has a FALSE on input <2>, a less detailed picklist is supplied, and only F:PICKINFO outputs <1>, <2>, and <5> are activated.

4. The integer on output <1> is the pick index, indicating which vector (in a vector list), character (in a character string), label (in a labels block), or parameter value (in a POLYNOMIAL or RATIONAL POLYNOMIAL curve) was picked. Vectors (or characters or labels) are assigned consecutive integer values in order of their appearance in the list (or string or labels block), beginning with 1.

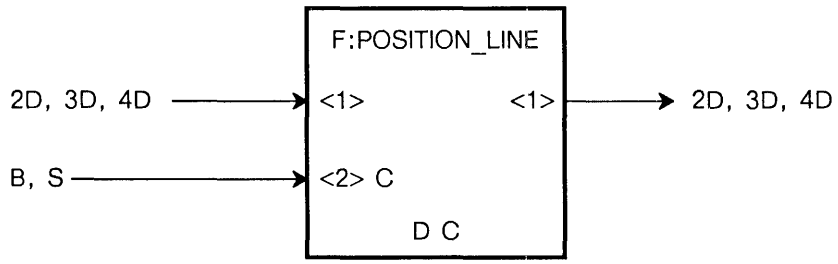
5. Output <2> is a string containing the requested pick IDs.

6. Output <3> is a 2D or 3D vector giving the coordinates of the intersection of the pickbox with the picked vector. Its data type depends on the data type of the picked vector (2D or 3D). Output <3> also reports the start location of a picked character string or label. (This output is supplied only for coordinate picklists.)
7. Output <4> gives the dimension (2 or 3) of the picked vector. (This output is supplied only for coordinate picklists.)
8. Output <5> is TRUE if coordinate picking information is being sent out, and FALSE otherwise. Output <5> is also false if coordinate picking is attempted on a character.
9. Output <6> gives the value of a polynomial parameter t (from 0 to 1, inclusive). This output is activated only for coordinate picklists resulting from picking a vector created by the POLYNOMIAL command or RATIONAL POLYNOMIAL command.
10. Output <7> is for an integer code specifying the data type of the object picked. The code may have values 1 through 8, corresponding to the following data types: (1) CHARACTERS; (2) 2D vector; (3) 3D vector (4) 2D POLYNOMIAL or RATIONAL POLYNOMIAL; (5) 3D POLYNOMIAL or RATIONAL POLYNOMIAL; (6) 2D BSPLINE or RATIONAL BSPLINE; (7) 3D BSPLINE or RATIONAL BSPLINE; (8) LABELS.
11. When output <8> is connected to input <1> of F:PRINT it causes F:PRINT to produce the name of the VECTOR_LIST, CHARACTERS, LABELS, BSPLINE, RATIONAL BSPLINE, POLYNOMIAL, or RATIONAL POLYNOMIAL command containing the picked vector.
12. If the command containing the picked vector is not named, a null is output at <8>.
13. Output <9> gives the physical screen coordinates.

F:POSITION_LINE

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts a 2D, 3D, or 4D vector on input <1>. A Boolean value on input <2> is used to assign a position (P) or line (L) to be associated with the vector. A string sent to input <2> consists of either a P or an L identifier. The vector, with the position/line condition specified by the Boolean value, is output on output <1>.

DESCRIPTION

INPUTS

<1> — any vector

<2> — Boolean value or string (constant)

OUTPUT

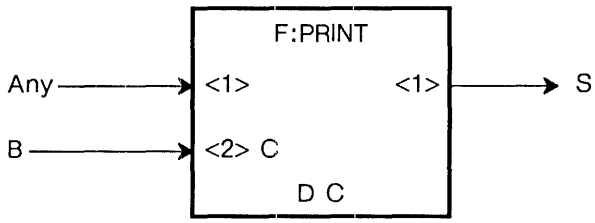
<1> — vector with P or L identifier

NOTE

A TRUE on input <2> causes a line (L) to be associated with the vector; a FALSE on input <2> causes a position (P) to be associated with the vector. The outputs from this function (vectors with position/line specifications) can only be applied to a vector list data node in a display structure. No function accepts such vectors as inputs.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Converts any data type to string format; that is, it performs an inverse of the operation that occurs when an ASCII string is input to the PS 390 and is converted to one of the data types.

DESCRIPTION

INPUTS

<1> — any message

<2> — Boolean value governing numeric format (constant)

OUTPUT

<1> — string

DEFAULT

The default for input <2> is FALSE, indicating decimal format.

NOTES

1. Screen coordinates, if passed to the function from F:PICKINFO, are added to the string output on <1>. Output <1> reports a pick in which coordinate picking information is given.

   For a vector declared in a VECTOR_LIST, the output string format is:

   ```
   <1><dimension><pick_x, pick_y, [pick_z]><index>
   <pick IDs><screen_x><screen_y>
   ```

For a vector within a polynomial curve the output string format is:

```
<2><dimension><pick_x, pick_y, [pick_z]><t>
<pick IDs><screen_x><screen_y>
```

2. Input <2> governs the format of real numbers and vectors (but not matrix elements) in the output string. When input <2> is FALSE, these values have the usual decimal format (e.g., '.001'). When input <2> is TRUE, these values are in exponential format (e.g. '1.000000E-3'). (Integers, on the other hand, are never in exponential format.) The output character string that results from each type of input follows:

Input Data Type	Output Character String
Boolean	'FALSE' or 'TRUE'.
Character	The same character that was input.
String	The same character string that was input.
Integer	The character representation of the integer; e.g., '129', '-107543'.
Real	A character representation of the real. e.g., '3.1416', '2.3E2' etc.

All vectors are preceded by a P (position), L (line), or V (no P or L) designation. ("X" in the following vector descriptions indicates P, L, or V.)

Input Data Type	Output Character String
2D Vector	Two real numbers separated by a comma; e.g., 'X 3.5,.0715'
3D Vector	Three real numbers separated by commas; e.g., 'X 3.1416,-275.012,3.5'
4D Vector	Four real numbers separated by commas; e.g., 'X 3.1416,-275.012,3.5,.0715'

Input Data Type	Output Character String
2x2 Matrix	Two 2D vectors (nine-digit precision, exponential format) separated by a space; e.g., '1.23456789E01, -2.56900187E-02 3.14159265E01, 2.71828183E01')
3x3 Matrix	Three 3D vectors (nine-digit precision, exponential format) separated by spaces.
4x4 Matrix	Four 4D vectors (nine-digit precision, exponential format) separated by spaces.
Pick list	The format of a picklist string depends on whether coordinate information was requested for the picklist (refer to F:PICKINFO and the PICK initial initial function instance) and, if it was requested, whether it was given. (For example, a vector in a character is not susceptible to standard coordinate picking.) All of these formats contain the clause <pick IDs>. This clause contains two things: first, a list of pick identifiers established in SET PICK ID,with the "closest" pick identifier first; second, a space followed by the name of the original data-definition command corresponding to the picked object. If this command is not named, neither a name nor a space follows the pick identifiers. If no coordinate picking information was requested (input <2> of the associated PICK function instance is FALSE), the output string has the format <index><pick IDs> for a vector in a declared vector list (including WITH PATTERN lists) or for a character in a string or label in a block, and < ><pick IDs> for a vector in a polynomial curve.

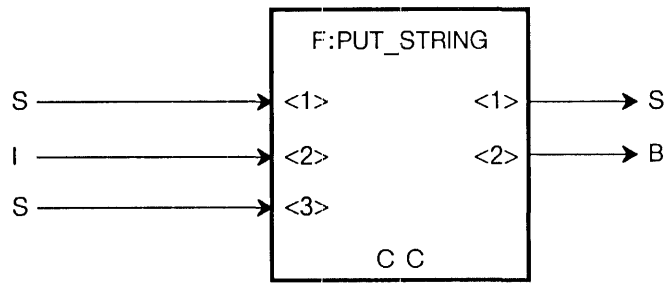
Input Data Type	Output Character String
Pick list(cont.)	If coordinate picking information was requested and given (i.e., if input <2> of the associated PICK is TRUE, and it was not a character vector), then the output string format is <1><dimension><pick_x, pick_y, [pick_z]> <index><pick IDs><screen_x><screen_y> for a vector in a declared vector list and <2><dimension><pick_x, pick_y,[pick_z]><t> <pick IDs><screen_x><screen_y> for a vector within a polynomial curve, where <dimension> and <t> are as defined for F:PICKINFO. For a character in a string the format is <3><dimension><start_x, start_y, start_z> <index><pick IDs> and for a label in a labels block, the format is <5><dimension><start_x, start_y, start_z> <index><pick IDs> If picked coordinates were requested but not given (i.e., input <2> of the associated PICK is TRUE and a vector in a character or in a polynomial curve was picked), the output string format is <3><index><pick IDs>

EXAMPLE

Refer to Helpful Hint 14 in Section *TT2*.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Replaces characters in the string on input <1> with the string on input <3>, starting at the position specified by the integer on input <2>. The resulting string may be longer than the original string if the string on input <3> overlaps. The Boolean value on output <2> is TRUE if the resulting string is the same length as the string on input <1>, and FALSE otherwise.

DESCRIPTION

INPUTS

<1> — string

<2> — starting location for replacing characters

<3> — replacement characters

OUTPUTS

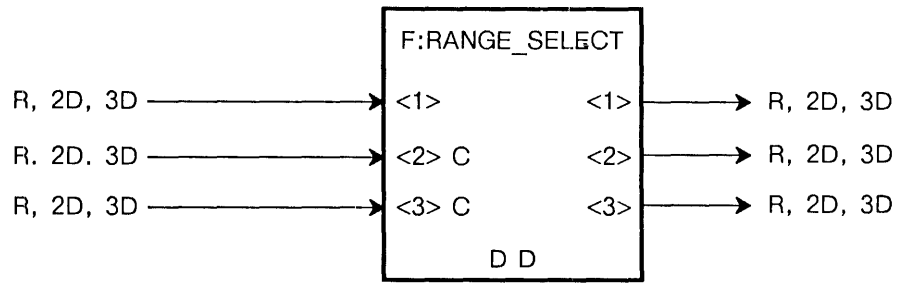
<1> — resulting string

<2> — TRUE = resulting string same length as the original, FALSE = resulting string longer than the original

F:RANGE_SELECT

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Compares the value on input <1> to the maximum and minimum on inputs <2> and <3> to determine whether the value is in range or not.

DESCRIPTION

INPUTS

<1> — value
<2> — maximum (constant)
<3> — minimum (constant)

OUTPUTS

<1> — in-range, normalized
<2> — in-range, unchanged
<3> — out-of-range, unchanged

NOTES

1. Accepts real number values or 2D or 3D vectors on all inputs. The data type must be the same on all inputs, as must the vector dimensions (that is, all vectors must be either 2D or 3D). The type of data output from the function is the same type that is input to the function.

2. The value on input <1> is compared to the constant maximum value on input <2> and the constant minimum value on input <3>.
3. If the value on input <1> is within the range defined by the minimum and maximum values (input <3> <= input <1> <= input <2>) then the value on input <1> is sent out on outputs <1> and <2>.
4. The value on output <1> is normalized to the maximum/minimum values of inputs <2> and <3>. The value on output <2> is identical to the input <1> value. If the value is in range, nothing is sent out on output <3>.
5. Data is normalized for output <1> by:

$$\text{normal X value} = \frac{X^1 - X^{min}}{\text{X range}} - 0.5$$

$$\text{normal Y value} = \frac{Y^1 - Y^{min}}{\text{Y range}} - 0.5$$

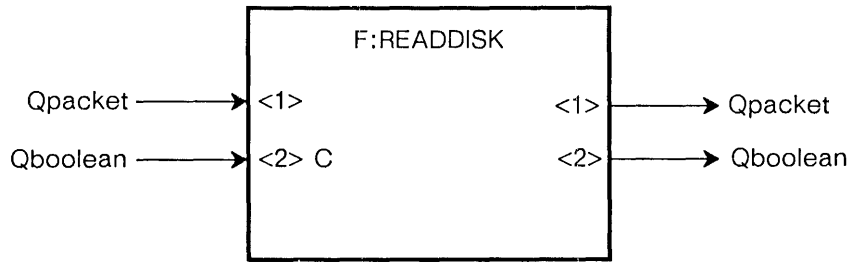
$$\text{normal Z value} = \frac{Z^1 - Z^{min}}{\text{Z range}} - 0.5$$

6. If the value on input <1> is not within range, it is output on output <3> unchanged.

F:READDISK

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

This function reads a file from the floppy disk and sends the data out <1> in Qpackets. Input <1> accepts a Qpacket of 1 to 8 characters specifying the name of the file to be read. All disk drives are searched for the file until found; if the file is not found, an error message is produced.

A TRUE on input <2> tells the function to delete the file after reading. Input <2> is a constant input queue and is initialized to FALSE.

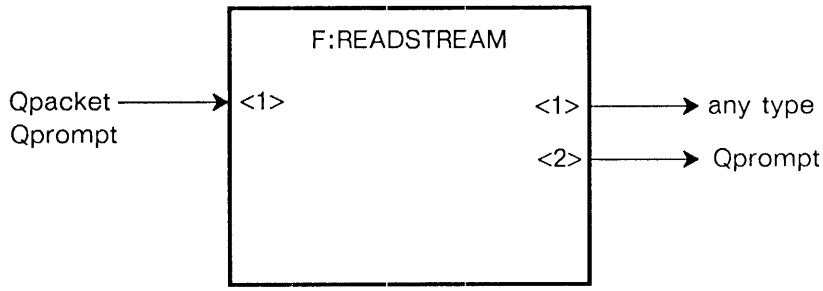
A TRUE is output from <2> when the file is found and read successfully. A FALSE is output when the file is not found.

NOTE

The file name sent on input <1> should not include the file extension. The file on the disk must have the extension ".DAT".

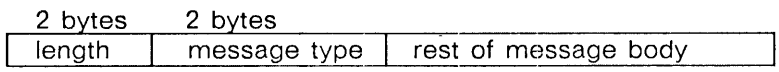
TYPE

Intrinsic User Function — Data Conversion



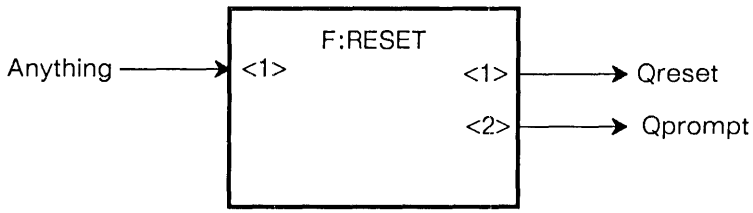
PURPOSE

This function converts an 8-bit stream into arbitrary messages. It takes two bytes as the count of information (including message type) and creates a message of that size with the bytes of information that follow it. The message format on input is:



TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

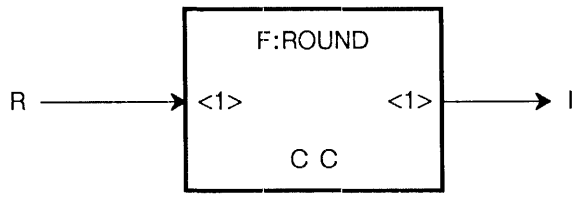
This function sends a Qreset out output <1> whenever it receives input other than a Qprompt or on input <1>. Qprompts are passed on through output <2>.

A Qreset purges a function and returns it to an initial state. Functions that respond to F:RESET are:

- F:CI(n)
- F:CIROUTE(n)
- F:DEMUX(n)
- F:DEPACKET
- F:GATHER_GENFCN
- F:PACKET
- F:RASTER
- F:RASTERSTREAM
- F:READSTREAM

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts a real number and outputs the nearest integral value.

DESCRIPTION

INPUT

<1> — real number

OUTPUT

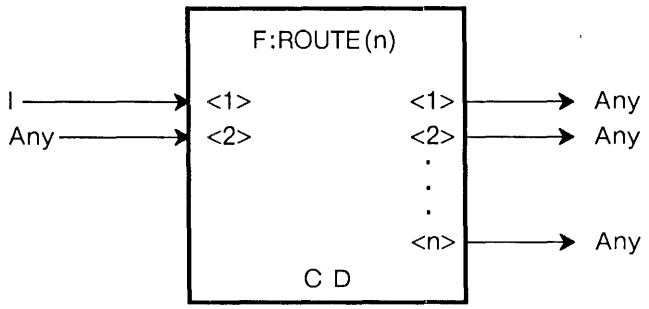
<1> — nearest integral value

NOTE

Values n to n.4999...9 are rounded to n; values n.5 to n.9999...9 are rounded to n+1. Values -n to -n.4999...9 are rounded to -n; values -n.5 to -n.999...9 are rounded to -n+(-1).

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Uses the integer on input <1> to route the message on input <2> to the output whose number matches the input <1> integer.

DESCRIPTION

INPUTS

<1> — number of selected output (1 through n)

<2> — any message

OUTPUTS

<1> — message on input <2>

•

•

•

<n> — message on input <2>

NOTE

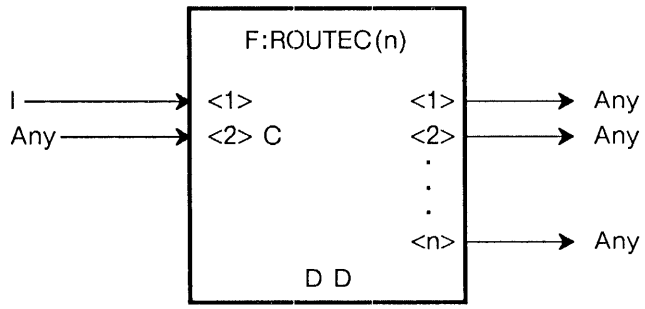
The message on input <2> may be of any data type. The "n" in the function name can be any integer from 2 to 127. If the integer on input <1> is not a number from 1 to n inclusive, then an error is detected and reported.

ASSOCIATED FUNCTIONS

F:ROUTEC(n), F:CROUTE(n)

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Uses the integer on input <1> to switch the message on input <2> to the output whose number matches the input <1> integer. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — number of selected output (1 through n)
<2> — any message (constant)

OUTPUTS

<1> — message on input <2>
•
•
•
<n> — message on input <2>

NOTE

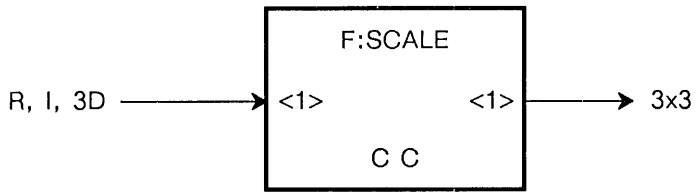
The message on input <2> may be of any data type. The "n" in the function name may be any integer from 2 to 127. If the integer on input <1> is not a number from 1 to n, inclusive, then the message on input <2> is held until a valid integer is received on input <1>.

ASSOCIATED FUNCTIONS

F:ROUTE(n), F:CROUTE(n)

TYPE

Intrinsic User Function — Object Transformation



PURPOSE

Accepts a real value, an integer, or a 3D vector. If a real value is input, the scaling factor represented by the real value is applied to X, Y, and Z. A 3x3 scaling matrix is output that may be used to update a scaling element of a display structure.

DESCRIPTION

INPUT

<1> — value

OUTPUT

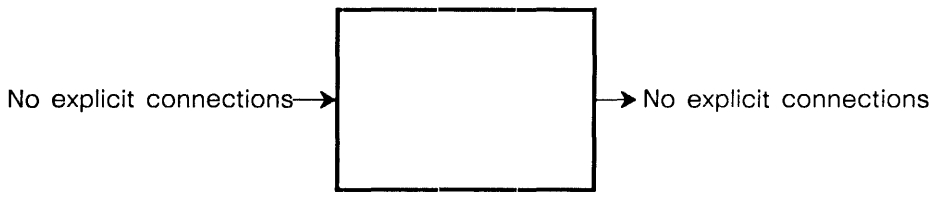
<1> — 3x3 scaling matrix

NOTE

If a 3D vector is input, the X component of the vector is the scaling factor for X, the Y component of the vector is the scaling factor for Y and the Z component of the the vector is the scaling factor for Z.

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

This function helps to protect the PS 390 screen from phosphor damage by slowly shifting the viewport in a way that is imperceptible to the user. The viewport moves right two line widths, up two line widths, left two line widths, and down two line widths, and repeats this cycle as long as F:SCREENSAVE is in effect. F:SCREENSAVE is on by default.

NOTES

1. Note that F:SCREENSAVE has no explicit inputs or outputs. The only way to use this function is to instance it when phosphor protection is desired and to delete the instance (using NIL) when it is not desired. To disable screen-saving, enter the command

   ```
   SCREENSAVE := NIL;
   ```

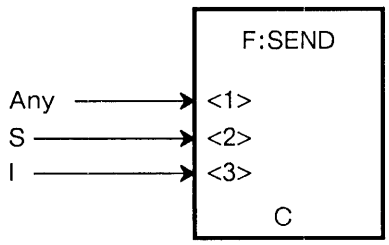
   To enable the screen-saving, enter the command

   ```
   SCREENSAVE := F:SCREENSAVE;
   ```

2. Screen-saving should be set to NIL before timed-exposure photographs of the PS 390 screen are taken.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

This is the function network equivalent of the SEND command. It allows you to send any valid data type to any named entity at any valid index.

DESCRIPTION

INPUTS

<1> — message sent

<2> — name of the destination node

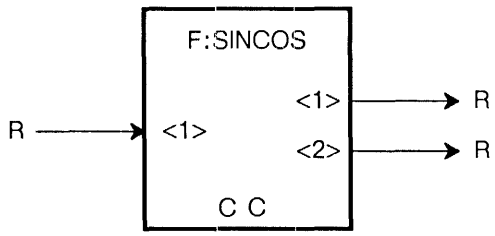
<3> — index into the destination node

NOTES

1. This function has no output.
2. Input <1> accepts special data types that most functions do not accept, such as the data type output by F:LABEL.
3. The SETUP CNESS command can be used to specify constant inputs as default values.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts a real number on input <1> which represents an angle in units of degrees. The sine of that angle is output as a real number on output <1>, and the cosine of that angle is output as a real number on output <2>.

DESCRIPTION

INPUT

<1> — angle

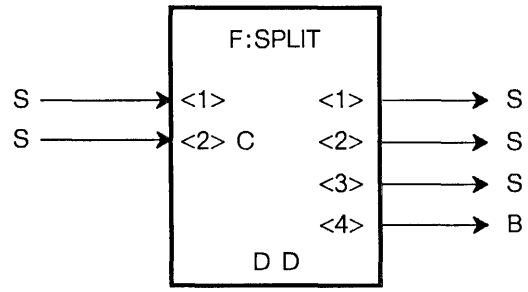
OUTPUTS

<1> — sine

<2> — cosine

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Accepts character strings on inputs <1> and <2>. The string on input <2> is a constant. When the string is received on input <1>, it is compared to the string on input <2> for an exact match.

DESCRIPTION

INPUTS

<1> — string

<2> — string (constant)

OUTPUTS

<1> — characters preceding match

<2> — matching characters

<3> — characters following match

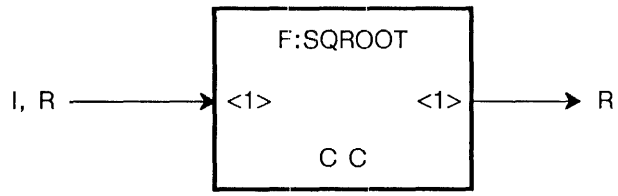
<4> — TRUE if matching inputs, FALSE otherwise

NOTES

1. If a match occurs, characters in the string on input <1> that precede the match are output on <1>. Matching characters are output on output <2>. Characters following the matching characters are output on output <3>. And a Boolean TRUE is output on output <4>.
2. If no match is found, nothing is output on outputs <1>, <2>, and <3>, and a Boolean FALSE is output on output <4>.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Extracts the square root of the real number or integer on input <1>.

DESCRIPTION

INPUT

<1> — real number or integer

OUTPUT

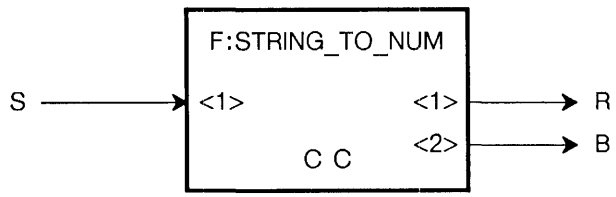
<1> — square root

NOTE

The output is always real. If the input is negative, the output is 0.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Outputs the value of a string of digits as a real number. If the function receives characters that cannot represent a number then an error message is generated.

DESCRIPTION

INPUT

<1> — string of digits

OUTPUTS

<1> — value of string on input <1>

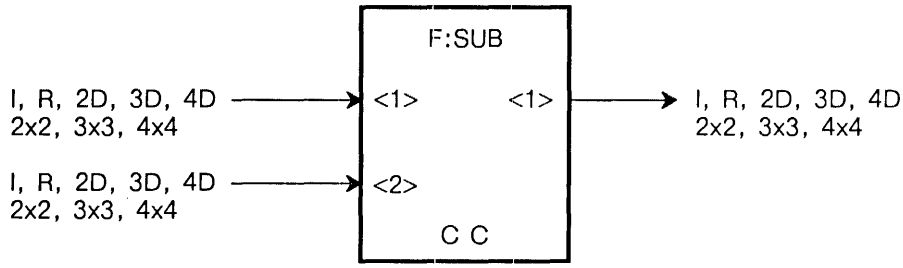
<2> — TRUE if the string received can be converted, FALSE otherwise

NOTE

A valid number can contain any or all of the following components: decimal point, 'E' expression, plus or minus sign, numerals.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs and produces an output that is the difference of the two inputs (input <2> is subtracted from input <1>).

DESCRIPTION

INPUTS

<1> — minuend

<2> — subtrahend

OUTPUT

<1> — difference

NOTE

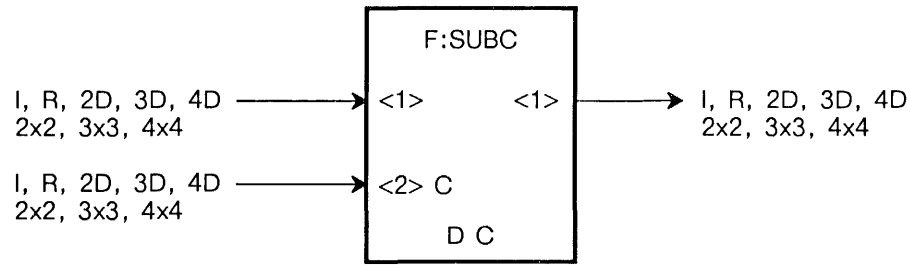
The two input values must be of the same data type (except a combination of real number and an integer is allowed); the output data type depends on the input data type. If a real number and an integer are input, a real number is output.

ASSOCIATED FUNCTIONS

F:SUBC, F:CSUB

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts two inputs and produces an output that is the difference of the two inputs (input <2> is subtracted from input <1>). Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — minuend

<2> — subtrahend (constant)

OUTPUT

<1> — difference

NOTE

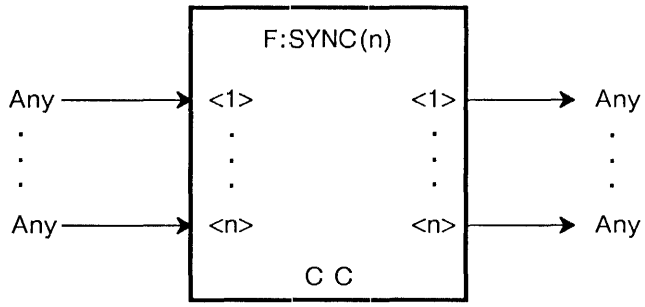
The two input values must be of the same data type (except a combination of real number and integer is allowed); the output data type depends on the input data type. If a real number and an integer are input, a real number is output.

ASSOCIATED FUNCTIONS

F:SUB, F:CSUB

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

Synchronizes the output of a specified number of messages. The number "n" may have any value from 2 to 127.

DESCRIPTION

INPUTS

<1> — any message

•
•
•

<n> — any message

OUTPUTS

<1> — any message

•
•
•

<n> — any message

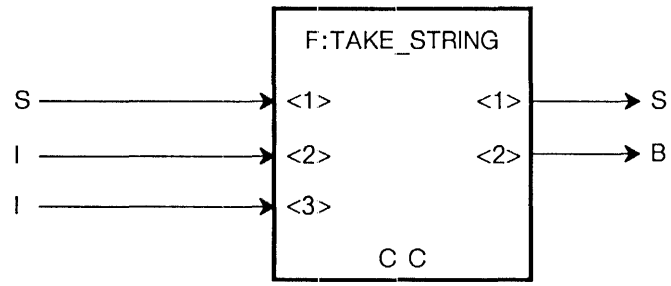
NOTES

1. F:SYNC(n) waits until a message is received on all of its "n" inputs, then sends the messages out; for example, F:SYNC(32) synchronizes 32 messages.

2. Usually, the outputs of an F:SYNC(n) function instance are connected to nodes in a display structure to assure that updates to displayed data are synchronized.
3. Outputs from F:SYNC(n) are effectively simultaneous. In fact, outputs are sequential (<1> through <n>) at a rapid rate.

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Outputs a string consisting of the number of characters specified on input <3> taken from the string on input <1>, starting at the position given on input <2>. A TRUE on output <2> means that there were enough characters left in the string. A FALSE means there were not enough characters, so the output string was truncated.

DESCRIPTION

INPUTS

<1> — string

<2> — starting position

<3> — number of characters to take

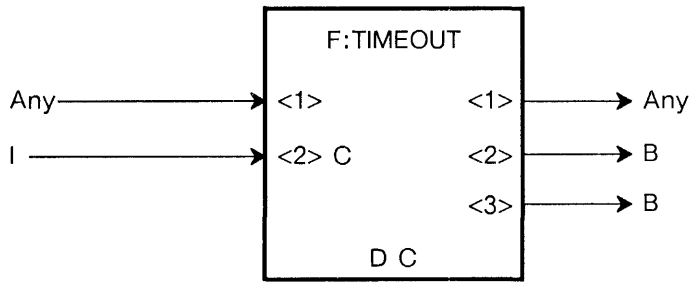
OUTPUTS

<1> — resulting string

<2> — TRUE = enough characters, FALSE = output string truncated

TYPE

Intrinsic User Function — Miscellaneous



PURPOSE

Provides the means to detect the occurrence of consecutive messages on input <1> within the time interval specified in centiseconds by the constant integer on input <2>.

DESCRIPTION

INPUTS

<1> — message on input <1>

<2> — time interval (constant)

OUTPUTS

<1> — any message

<2> — TRUE = timeout, FALSE = no timeout

<3> — logical complement of output <2>

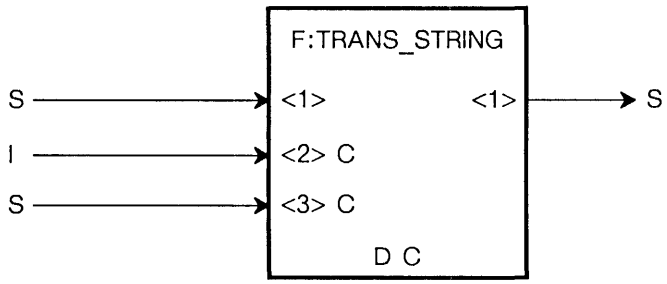
NOTES

1. Once the first message is received on input <1>, the subsequent message must be received in the duration specified on input <2> in order to be passed through the function. Then the third message must be received within that specified duration after the second message, and so on.

2. The first message to input <1> serves only to start the timeout measurement, and never generates an output.
3. If any subsequent messages are received at input <1> within the time interval specified on input <2>, only the last message is sent on output <1> at the end of the interval; all intervening messages are discarded.
4. If a message on input <1> is not received within the specified time, the Boolean value on output <2> is TRUE. If a message on input <1> is received within the interval, the Boolean value on output <2> is FALSE. Output <3> is the complement of output <2>.
5. This function is especially useful to determine a data tablet stylus out-of-range condition. If the message from the data tablet stylus is connected to input <1> of this function and an appropriate duration is specified on input <2>, then the inputs from the data tablet will be passed through the function until the duration is exceeded.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Translates the string on input <1> into the output string using the string on input <3> as a translation table. The integer on input <2> is the beginning place (i.e., the ASCII decimal equivalent or ORD) of the first character to be translated. Inputs <2> and <3> are constants.

DESCRIPTION

INPUTS

<1> — string

<2> — first character to be translated (constant)

<3> — translation table (constant)

OUTPUT

<1> — translated string

NOTE

The upper limit of the number of characters to translate is the length of the string on input <3>.

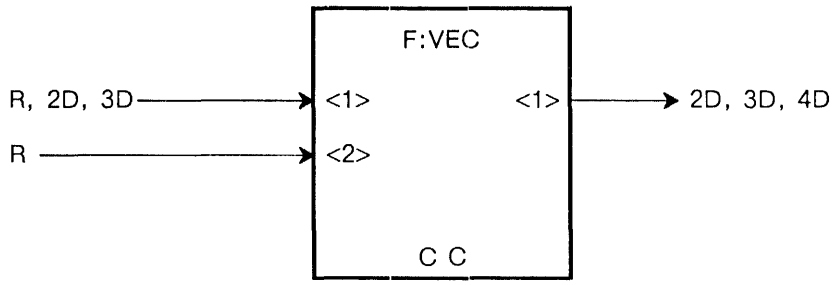
EXAMPLE

```
SEND 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' TO <3>Trans_String;
SEND FIX(97) TO <2>Trans_String; {the ASCII equivalent of 'a'}
SEND 'abcdefghijklmnopqrstuvwxyz' TO <1>Trans_String;
```

The lower case letters sent to input <1> will be translated to uppercase on output <1>.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts two real numbers and outputs a 2D vector, accepts a 2D vector and a real number and outputs a 3D vector, or accepts a 3D vector and a real number and outputs a 4D vector.

DESCRIPTION

INPUTS

<1> — real number, 2D, or 3D vector

<2> — real number

OUTPUT

<1> — vector consisting of the value on input <1> with the real number on input <2> appended

NOTE

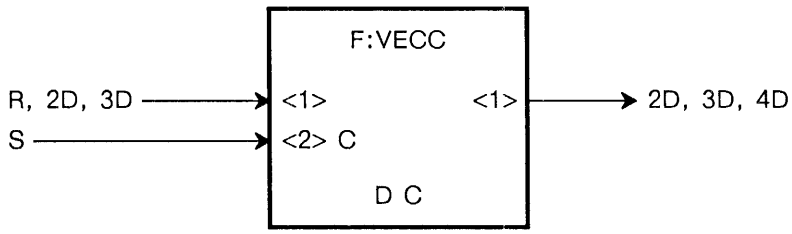
The output vector is the real number or vector from input <1> with the real number from input <2> appended as the last vector component.

ASSOCIATED FUNCTIONS

F:VECC, F:CVEC

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts two real numbers and outputs a 2D vector, accepts a 2D vector and a real number and outputs a 3D vector, or accepts a 3D vector and a real number and outputs a 4D vector. Input <2> is a constant.

DESCRIPTION

INPUTS

<1> — real number, 2D, or 3D vector

<2> — real number (constant)

OUTPUT

<1> — vector consisting of the value on input <1> with the real number on input <2> appended

NOTE

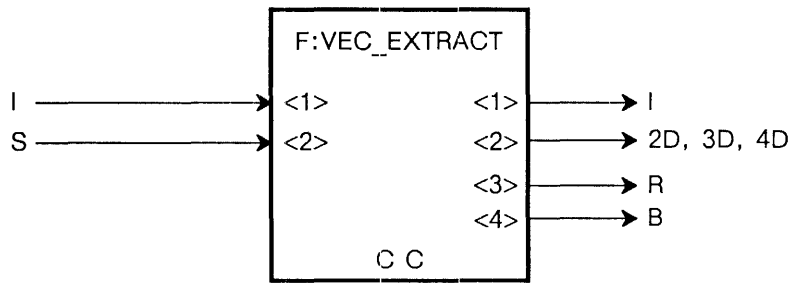
The output vector is the real number or vector from input <1> with the real number from input <2> appended as the last vector component.

ASSOCIATED FUNCTIONS

F:VEC, F:CVEC

TYPE

Intrinsic User Function — Data Selection and Manipulation



PURPOSE

Extracts information about a vector in a vector list node given an index into the vector list on input <1> and the name of the vector list node on input <2>.

DESCRIPTION

INPUTS

<1> — index of the vector in question

<2> — name of the vector list node

OUTPUTS

<1> — data type

<2> — the vector in question

<3> — intensity

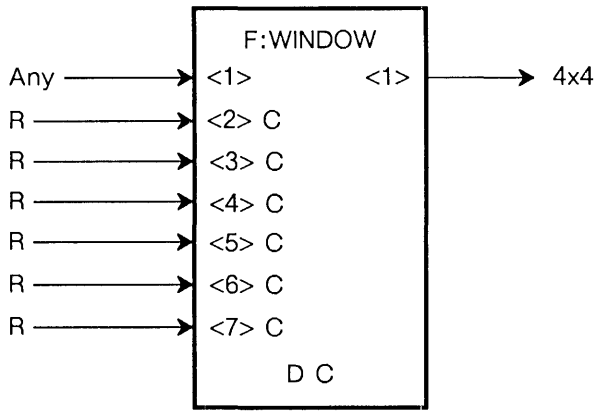
<4> — TRUE = line, FALSE = position

NOTE

The integer on output <1> is the same as would be sent from output <7> of F:PICKINFO.

TYPE

Intrinsic User Function — Viewing Transformation



PURPOSE

This is the functional counterpart of the WINDOW command. The window matrix that results from this function defines a viewing area for orthographic views (parallel projections) of objects.

DESCRIPTION

INPUTS

<1> — trigger
<2> — X minimum (constant)
<3> — X maximum (constant)
<4> — Y minimum (constant)
<5> — Y maximum (constant)
<6> — Z minimum (constant)
<7> — Z maximum (constant)

OUTPUT

<1> — 4x4 matrix

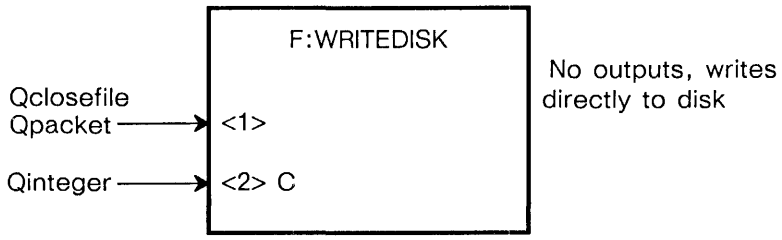
NOTES

6. F:WINDOW accepts any message on input <1> to trigger the function and constant real values on inputs <2> through <7>. These real values define the boundaries of a three-dimensional rectangular volume within which objects can be viewed in parallel projection (i.e. no perspective is imposed).

7. This volume is defined by expressing a rectangle in terms of Xmin (input <2>), Xmax (input <3>), Ymin (input <4>), and Ymax (input <5>). The rectangle is then extended into a three dimensional volume by specifying Zmin (input <5>) and Zmax (input <7>).

TYPE

Intrinsic User Function — Data Selection and Manipulation

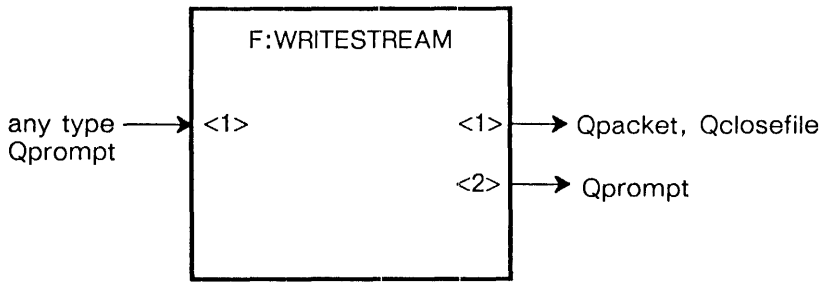


PURPOSE

This function writes its input messages (Qpackets) to a file on the mini-floppy diskette. The message contents are collected in buffers which are stored on its private queue. When a message of type Qclosefile is received, the data in the buffers is written to disk on the drive specified on input <2> and with the file name (with ".DAT" file extension) given in the message Qclosefile. A Qclosefile message can only be obtained by sending a CLOSE "filename" command to a F:CHOP function whose output is connected to the F:WRITEDISK function.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

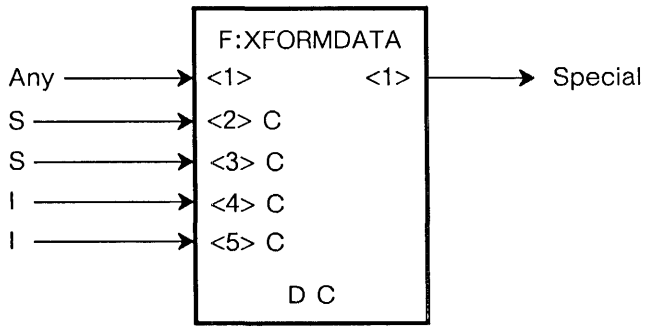
This function takes any message and turns it into a stream of bytes in a Qpacket (except for Qprompts and Qclosefiles). Qprompts are passed on through to output <2> and Qclosefiles are passed onto output <1>. This is used to create binary data files on a floppy diskette. Normally, this function receives input from a chop/parse function and sends output to a WRITEDISK function.

The resulting Qpackets contain:

2 bytes	2 bytes	
length	message type	rest of message body

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Sends transformed data (either a vector list or a 4x4 matrix) to a specified destination (e.g., the host, a printer, or the screen).

DESCRIPTION

INPUTS

<1> — any message

<2> — name of XFORM node (constant)

<3> — name of destination object (constant)

<4> — destination vector index (constant)

<5> — number of vectors (constant)

OUTPUT

<1> — special data type used as input to F:LIST, F:PRINT, or SURFACE_Rendering or SOLID_Rendering operation node.

DEFAULT

Default for input <4> is 1, default for input <5> is 2048.

NOTES

1. Input <1> is a trigger for F:XFORMDATA. This input would typically be connected to a function button, either directly or via F:SYNC(2), allowing transformed data to be requested easily.

2. Input <2> is a string or matrix containing the name of the XFORM command in the display structure (either XFORM MATRIX or XFORM VECtor). By referring to an XFORM command, this input indirectly specifies the object whose transformed data is to be sent. If the string names something other than an XFORM command, an error message is displayed. If the string names a node which does not exist, an error message is sent and the message is removed from input <2>.

3. Input <3> is a string containing the name to be associated with the transformed vectors. The name need not be previously defined. If this input does not contain a valid string, the transformed matrix or vectors will be created without a name (an acceptable situation unless the transformed vectors need to be referenced or displayed.) The transformed vector list can be displayed or modified, provided a name is given on this input. The transformation matrix cannot be used, however, so naming and sending it to input <3> is not useful.

4. Input <4> is an integer index specifying the place in a vector list at which the PS 390 is to start returning transformed data. This input is only used when the command name at input <2> represents an XFORM VECTOR command (not an XFORM MATRIX command). The default value is 1.

5. Input <5> is an integer number of consecutive vectors for which transformed data is to be returned, starting at the vector specified at input <4>. This input is only used when the command name at input <2> represents an XFORM VECTOR command (not an XFORM MATRIX command). No more than 2048 consecutive vectors may be returned. The default value is 2048.

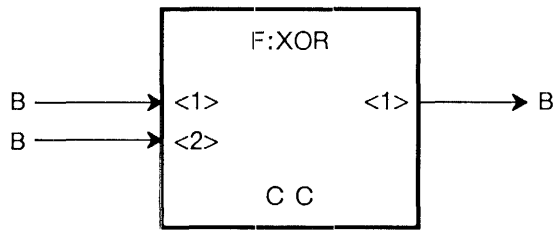
6. Output <1> contains the transformed data in a format which can be accepted by input <1> of F:LIST or F:PRINT and inputs <3> and <4> of a SURFACE_Rendering or SOLID_Rendering operation node. F:LIST prints out the data in ASCII format - either a PS 390 VECTOR_LIST command or a PS 390 MATRIX_4X4 command, depending on whether the command named at input <2> was an XFORM VECTOR or an XFORM MATRIX.
7. F:XFORMDATA is used in connection with rendering lines and spheres on the PS 390 display.

EXAMPLE

Refer to Helpful Hints 4 and 14 in Section *TT2*.

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts Boolean values on inputs <1> and <2>, performs an exclusive-OR function on the values, and outputs the result as a Boolean value. That is, if the Boolean values on both inputs are the same, the output is FALSE; if the Boolean values on the inputs are different, the output is TRUE.

DESCRIPTION

INPUTS

<1> — Boolean value

<2> — Boolean value

OUTPUT

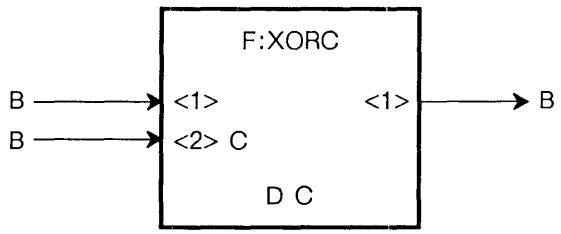
<1> — exclusive OR of inputs

ASSOCIATED FUNCTION

F:XORC

TYPE

Intrinsic User Function — Arithmetic and Logical



PURPOSE

Accepts Boolean values on inputs <1> and <2>, performs an exclusive-OR function on the values, and outputs the result as a Boolean value. Unlike F:XOR, for which both inputs are active, F:XORC input <2> is a constant. If the Boolean values on both inputs are the same, the output is FALSE; if the Boolean values on the inputs are different, the output is TRUE.

DESCRIPTION

INPUTS

<1> — Boolean value

<2> — Boolean value (constant)

OUTPUT

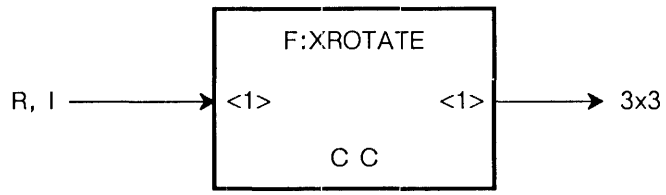
<1> — exclusive OR of inputs

ASSOCIATED FUNCTION

F:XOR

TYPE

Intrinsic User Function — Object Transformation



PURPOSE

Accepts a real value or an integer that specifies the number of degrees about the X axis that the rotation matrix generated by the function is to represent.

DESCRIPTION

INPUT

<1> — degrees of rotation in X

OUTPUT

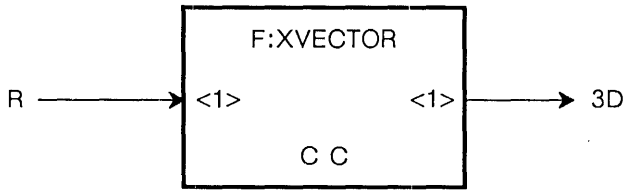
<1> — 3x3 rotation matrix

NOTE

The 3x3 rotation matrix which is output may be used to update a rotation node in a display structure.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts a real number on input <1> and outputs a 3D vector.

DESCRIPTION

INPUT

<1> — real number

OUTPUT

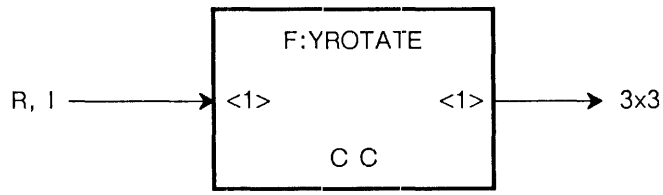
<1> — 3D vector

NOTE

In the 3D vector which is output, X is equal to the input real, and Y and Z are 0. For example, if 3 were input, the 3D vector output would be 3,0,0.

TYPE

Intrinsic User Function — Object Transformation



PURPOSE

Accepts a real value or an integer that specifies the number of degrees about the Y axis that the rotation matrix generated by the function is to represent.

DESCRIPTION

INPUT

<1> — degrees of rotation in Y

OUTPUT

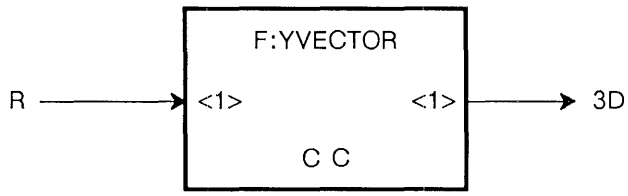
<1> — 3x3 rotation matrix

NOTE

The 3x3 rotation matrix that is output may be used to update a rotation node in a display structure.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts a real number on input <1> and outputs a 3D vector.

DESCRIPTION

INPUT

<1> — real number

OUTPUT

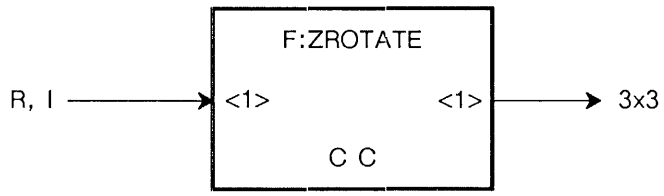
<1> — 3D vector

NOTE

In the 3D vector which is output, Y is equal to the input real, and X and Z are 0. For example, if 4 were input, the 3D vector output would be 0,4,0.

TYPE

Intrinsic User Function — Object Transformation



PURPOSE

Accepts a real value or an integer that specifies the number of degrees about the Z axis that the rotation matrix generated by the function is to represent.

DESCRIPTION

INPUT

<1> — degrees of rotation in Z

OUTPUT

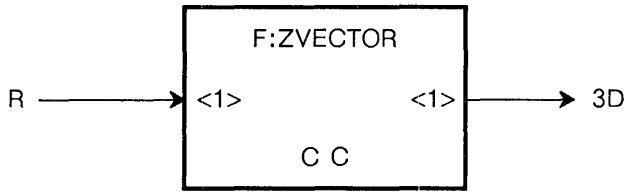
<1> — 3x3 rotation matrix

NOTE

The 3x3 rotation matrix that is output may be used to update a rotation node in a display structure.

TYPE

Intrinsic User Function — Data Conversion



PURPOSE

Accepts a real number on input <1> and outputs a 3D vector.

DESCRIPTION

INPUT

<1> — real number

OUTPUT

<1> — 3D vector

NOTE

In the 3D vector which is output, Z is equal to the input real, and X and Y are 0. For example, if 5 were input, the 3D vector output would be 0,0,5.

7. Intrinsic System Functions

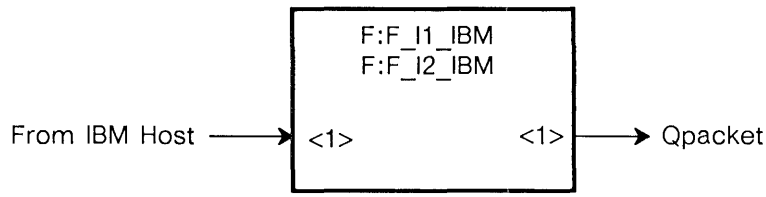
Following is a summary of the Intrinsic System Functions. The functions are ordered alphabetically on a letter-by-letter basis.

The following information, where relevant, is given for each function:

- Name
- Type - Category
- Purpose
- Description of inputs and outputs
- Defaults
- Notes
- Associated functions
- Examples

TYPE

Intrinsic System Function — Data Selection and Manipulation

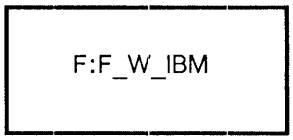


PURPOSE

F:F_I1_IBM, F:F_I2_IBM output packets of characters received from an IBM host on output <1>.

TYPE

Intrinsic System Function — Miscellaneous



PURPOSE

F:F_W_IBM wakes up every frame and disposes of packets which were passed to F:F_O1_IBM, F:F_O2_IBM, F:F_K1_IBM, AND F:F_K2_IBM, as well as waking up F:F_ I1_IBM AND F:F_I2_IBM when data has been received from an IBM host.

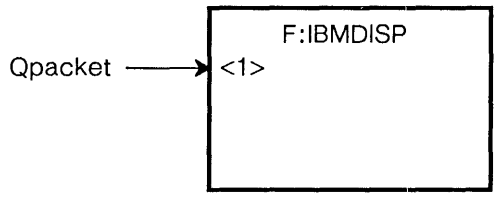
It also checks to see if the GPIO board has timed out and displays the indicator character

'G'

on the terminal emulator screen if a timeout has occurred.

TYPE

Intrinsic System Function — Miscellaneous

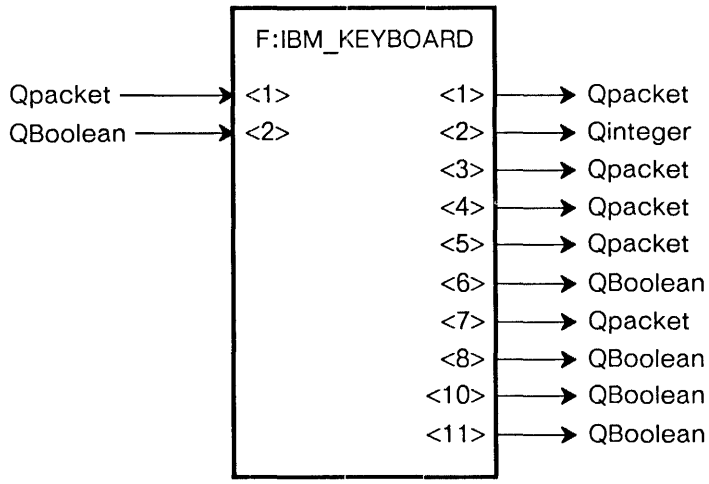


PURPOSE

F:IBMDISP accepts packets of ASCII characters on input <1> and either inserts their equivalent IBM screen code into the local screen buffer used by the Command mode of terminal emulator or causes the cursor position to be adjusted in the case of a carriage return, a line feed, or a back space.

TYPE

Intrinsic System Function — Data Selection and Manipulation



PURPOSE

F:IBM_KEYBOARD accepts character packets from the keyboard on input <1> and based on the mode selected by the mode keys (either the LINE LOCAL key or the HOST, LOCAL and COMMAND keys, depending on the type of keyboard used), outputs packets for use by the function network, the line editor, or an IBM host. Packets of characters for the function KEYBOARD are output on output <1>. Qintegers to be sent to the function FKEYS are output on output <2>. Qpackets of characters to be sent to the function SPECKEYS are output on output <3>. Qpackets of characters for the line editor are output on output <4>. Qpackets of IBM scan codes for an IBM host are output on output <5>. A QBOOLEAN TRUE used to trigger the hardcopy functions is output on either output <6>, output <10>, or output <11>, based on the mode of the keyboard.

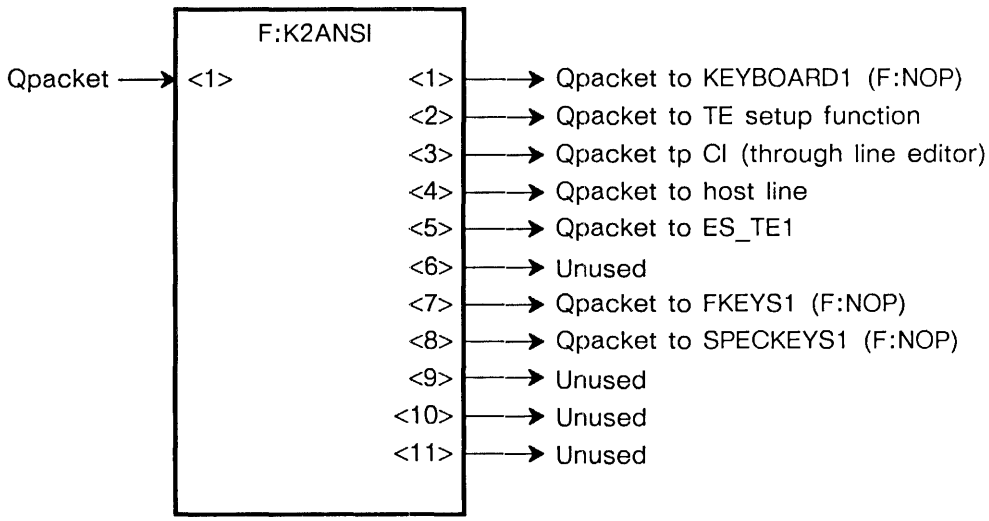
NOTES

1. A TRUE used to trigger the loading of the IBM 3250 function network is output on output <7> when IBM 3250 mode is selected while in SETUP mode.

2. A TRUE used to trigger the deletion of the IBM 3250 function network is output on output <8> when the PS 390 mode is selected while in SETUP mode.
3. Input <2> accepts a Boolean value that indicates which type of keyboard is being used: TRUE = IBM style keyboard; FALSE = VT100 style keyboard.

TYPE

Intrinsic System Function — Data Selection and Manipulation

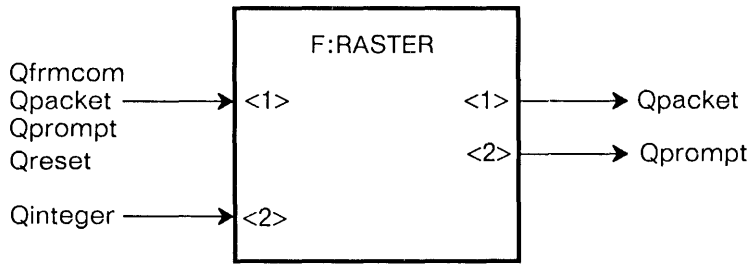


PURPOSE

This function takes the stream of raw bytes from the keyboard and distributes them to output queues, translating to ANSI control sequences if necessary; toggles graphics and terminal emulator displays.

TYPE

Intrinsic System Function — Data Selection and Manipulation



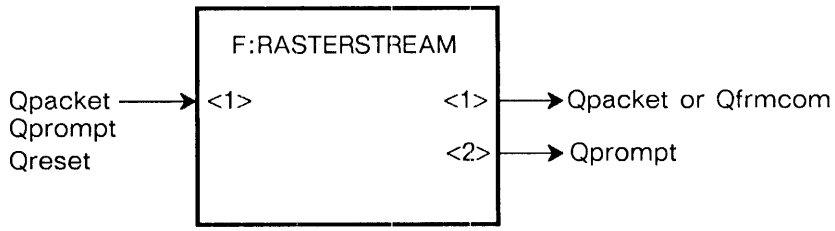
PURPOSE

The function passes the data from F:RASTERSTREAM to the ACP and disposes of the information after the ACP has completed its task. When information is read back from the raster, it prefixes each set of data with a 2 byte (16 bit) byte count and then sends two bytes of 0 after the completion of all information requests. Qfrmcom either sets the mode for interpretation of subsequent Qpackets received, or requests the return of raster or color table data. Qprompt is passed to output <2>, Qreset is ignored.

Input <2> is used to specify packet size.

TYPE

Intrinsic System Function — Data Selection and Manipulation

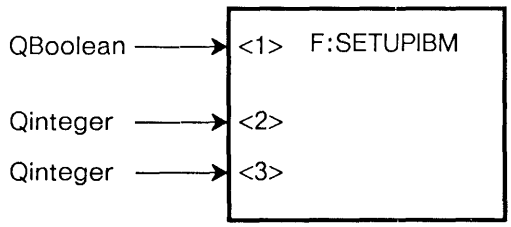


PURPOSE

The function takes streams of bytes (usually from the host) and either makes a Qpacket or Qfrmcom (special data type used by F:RASTER) from them. It works in a count mode where it takes 2 bytes (16 bits) to specify the count. If the count = 0, then a frame command is to be generated and the next two bytes are the value. If the count > 0, then a Qpacket with that many bytes is to be generated. A Qprompt is transmitted through to output <2>. A Qreset causes any data that may be stored to be disposed and to expect a new count to be received next.

TYPE

Intrinsic System Function — Miscellaneous



PURPOSE

F:SETUPIBM is used to change the parameters used by the IBM communications. Input <1> accepts an integer that specifies the maximum number of packets that can be in the pool of empty input packets.

DESCRIPTION

INPUTS

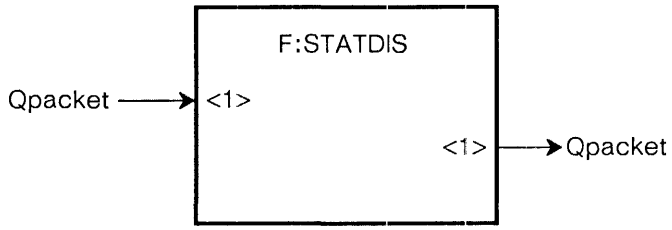
<1> — triggers the function.

<2> — specifies the number of empty I/O input packets that are to be maintained in the I/O input pool.

<3> — specifies the device address when an IBM 3250 interface is being used.

TYPE

Intrinsic System Function — Miscellaneous

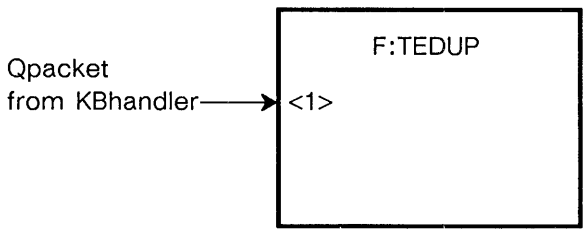


PURPOSE

This function causes messages to be displayed on the memory_display line of the PS 390 display. Output <1> is connected at system initialization for the function labels. Whenever this function receives a bell character, CHAR(7), it sends the character out output <1>.

TYPE

Intrinsic System Function — Miscellaneous

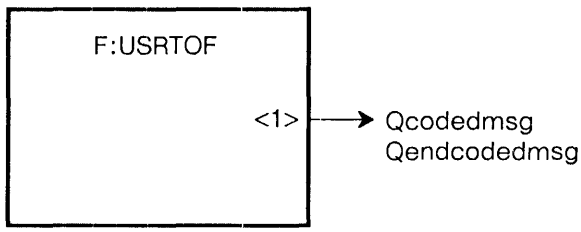


PURPOSE

The function allows users to change the VT100 terminal emulator configurable characteristics at runtime by depressing the SETUP key on the PS 390 keyboard.

TYPE

Intrinsic System Function — Miscellaneous



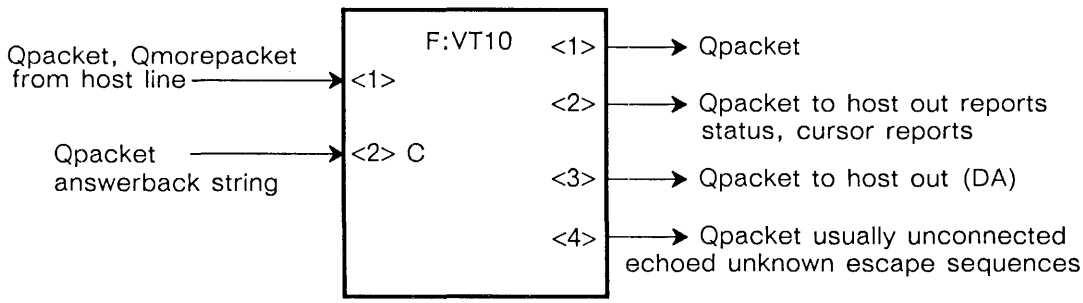
PURPOSE

This is one of the two functions that handle ACP timeouts. When the ACP has been processing a frame for more than the specified time limit (one second, except when the PS 390 is performing a viewing operation) this function is executed. It removes all user objects being displayed and sends out a message indicating the number of timeouts since boot time.

Timeouts usually occur because a recursive structure has been displayed.

TYPE

Intrinsic System Function — Data Selection and Manipulation



PURPOSE

This is the VT100 terminal emulator function. It receives input from the host, the line editor, and other sources on input <1>. The primary task of this function is to route the input to the PS 390 display in an appropriate manner.

Input <2> defines the answerback string that is sent to the host when this function receives an ENQ.

Output <1> is used to make the expected "beep" on receipt of a ↑G (the beeper is in the keyboard). Output <2> sends data back to the host when the function receives command sequences, such as cursor position and terminal ID (I'm a VT-100). Output <3> is used to send the correct control sequence back to the host that identifies the terminal. Output <4> is an aid for debugging and development. It sends out all command sequences that are received, but unknown. Normally output <4> is not hooked up. It can be used to discover what kind of sequences a host program might be sending (that the terminal emulator cannot interpret) by hooking the output to a function like MESSAGE_DISPLAY.

EXAMPLE

Refer to Helpful Hint 14 in Section *TT2*.

Appendix A
Intrinsic Functions by Category

Classification of Functions

Functions can be classified by the operations they perform. The PS 390 functions can be classified into the following nine categories:

- Arithmetic and Logical
  There functions perform all arithmetic operations (add, subtract, divide, multiply, square root, sine, and cosine) and logical operations (and, or, exclusive-or, and complement).
- Character Transformation
  These functions are used to interactively position, rotate, and scale text.
- Comparison
  These functions test whether values are greater than, less than, equal to, not equal to, greater than or equal to, and less than or equal to other values.
- Data Conversion
  These functions combine vectors into matrices, extract vectors from matrices, form vectors from real numbers, round or truncate real numbers to integers, float integers to equivalent real numbers, make printable characters, and convert appropriate character strings to a string of integers.
- Data Input and Output
  These functions set up and control the interactive devices (dials, function buttons, function keys, data tablet, and keyboard) and output values to the optional LED labels that several of the devices have. These functions are also used for communications with the host computer.
- Data Selection and Manipulation
  These functions are used to selectively switch functions, choose outputs, and route data.

- Miscellaneous
  These functions are used to set up and control clocking, timing, and synchronizing operations.

- Object Transformation
  These functions connect to modeling operation nodes in display structures to interactively rotate, translate, and scale objects.

- Viewing Transformation
  These functions connect to viewing operation nodes in display structures to interactively change line-of-sight, window size, and viewing angle.

Intrinsic User Functions

Arithmetic and Logical

- F:ACCUMULATE
- F:ADD
- F:ADDC
- F:AND
- F:ANDC
- F:AVERAGE
- F:CDIV
- F:CMUL
- F:CSUB
- F:DIV
- F:DIVC
- F:MOD
- F:MODC
- F:MUL
- F:MULC
- F:NOT
- F:OR
- F:ORC
- F:ROUND
- F:SINCOS
- F:SQROOT
- F:SUB
- F:SUBC
- F:XOR
- F:XORC

Character Transformation

- F:CROTATE
- F:CSCALE

Comparison

- F:CGE
- F:CGT
- F:CLE
- F:CLT
- F:COMP_STRING
- F:EQ
- F:EQC
- F:GE
- F:GEC
- F:GT
- F:GTC
- F:LE
- F:LEC
- F:LT
- F:LTC
- F:NE
- F:NEC

Data Conversion

- F:ALLOW_VECNORM
- F:CEILING
- F:CHANGEQTYPE
- F:CHARCONVERT
- F:CHOP
- F:CVEC
- F:CVT6TO8
- F:CVT8TO6
- F:CVTASCTOIBM
- F:CVTIBMTOASC
- F:FIX
- F:FLOAT
- F:LIST
- F:MAKEPACKET
- F:MATRIX2
- F:MATRIX3
- F:MATRIX4
- F:NPRT_PRT
- F:PARTS
- F:PICKINFO
- F:PRINT
- F:READSTREAM
- F:STRING_TO_NUM
- F:TRANS_STRING
- F:VEC
- F:VECC
- F:WRITESTREAM
- F:XFORMDATA
- F:XVECTOR
- F:YVECTOR
- F:ZVECTOR

Data Selection and Manipulation

- F:ATSCALE
- F:BOOLEAN_CHOOSE
- F:BROUTE
- F:BROUTEC
- F:CBROUTE
- F:CCONCATENATE
- F:CHARMASK
- F:CIROUTE(n)
- F:CONCATENATE
- F:CONCATENATEC
- F:CONCATXDATA(n)
- F:CONSTANT
- F:CROUTE(n)
- F:DELTA
- F:DEMUX(n)
- F:DEPACKET
- F:FCNSTRIP
- F:FIND_STRING
- F:GATHER_STRING
- F:INPUTS_CHOOSE(n)
- F:LABEL
- F:LBL_EXTRACT
- F:LENGTH_STRING
- F:LIMIT
- F:LINEEDITOR
- F:MCAT_STRING(n)
- F:MINMAX(n)
- F:MUX
- F:PACKET
- F:PASSTHRU(n)
- F:POSITION_LINE
- F:PUT_STRING
- F:RANGE_SELECT
- F:READDISK
- F:RESET
- F:ROUTE(n)
- F:ROUTEC(n)
- F:SEND
- F:SPLIT
- F:TAKE_STRING
- F:VEC_EXTRACT
- F:WRITEDISK

Miscellaneous

- F:CI(n)
- F:CLCSECONDS
- F:CLFRAMES
- F:CLTICKS
- F:EDGE_DETECT
- F:FETCH
- F:GATHER_GENFCN
- F:HOLDMESSAGE

- F:NOP
- F:SCREENSAVE
- F:SYNC(n)
- F:TIMEOUT
- F:USRTOF

Object Transformation

- F:DSCALE
- F:DXROTATE
- F:DYROTATE
- F:DZROTATE
- F:SCALE
- F:XROTATE
- F:YROTATE
- F:ZROTATE

Viewing Transformation

- F:FOV
- F:LOOKAT
- F:LOOKFROM
- F:WINDOW

Intrinsic System Functions

Data Selection and Manipulation

- F:F_I1_IBM
- F:F_I2_IBM
- F:IBM_KEYBOARD
- F:K2ANSI
- F:RASTER
- F:RASTERSTREAM
- F:VT10

Miscellaneous

- F:F_W_IBM
- F:IBMDISP
- F:SETUPIBM
- F:STATDIS
- F:TEDUP
- F:USRTOF

ASCII Character Code Set

Decimal Value	ASCII Character	Decimal Value	ASCII Character	Decimal Value	ASCII Character
0	NUL	44	'	88	X
1	SOH	45	-	89	Y
2	STX	46	.	90	Z
3	ETX	47	/	91	[
4	EOT	48	0	92	\
5	ENQ	49	1	93	]
6	ACK	50	2	94	↑ or ^
7	BEL	51	3	95	← or _
8	BS	52	4	96	`
9	HT	53	5	97	a
10	LF	54	6	98	b
11	VT	55	7	99	c
12	FF	56	8	100	d
13	CR	57	9	101	e
14	SO	58	:	102	f
15	SI	59	;	103	g
16	DLE	60	<	104	h
17	DC1	61	=	105	i
18	DC2	62	>	106	j
19	DC3	63	?	107	k
20	DC4	64	@	108	l
21	NAK	65	A	109	m
22	SYN	66	B	110	n
23	ETB	67	C	111	o
24	CAN	68	D	112	p
25	EM	69	E	113	q
26	SUB	70	F	114	r
27	ESC or ALT	71	G	115	s
28	FS	72	H	116	t
29	GS	73	I	117	u
30	RS	74	L	118	v
31	VS	75	K	119	w
32	SP	76	L	120	x
33	!	77	M	121	v
34	"	78	N	122	z
35	#	79	O	123	{
36	$	80	P	124	|
37	%	81	Q	125	}
38	&	82	R	126	~ TILDE
39	´	83	S	127	Rubout or DEL
40	(	84	T		
41	)	85	U		
42	*	86	V		
43	+	87	W		

RM3. INITIAL FUNCTION INSTANCES

RM3. INITIAL FUNCTION INSTANCES

CONTENTS

BUTTONSIN 2
CLEAR_LABELS 3
DIALS 4
DLABEL1...DLABEL8 6
DSET1...DSET8 8
ERROR 10
FKEYS 11
FLABEL0 12
FLABEL1...FLABEL12 14
HOST_MESSAGE
HOST_MESSAGEB 16
HOSTOUT 18
INFORMATION 19
KEYBOARD 20
MEMORY_ALERT 21
MEMORY_MONITOR 23
MESSAGE_DISPLAY 25
MOUSEIN 26
OFFBUTTONLIGHTS 29
ONBUTTONLIGHTS 30
PICK 31
PS390ENV 35
SHADINGENVIRONMENT 37
SPECKEYS 46
TABLETIN 47
TABLETOUT 50
TECOLOR 52
WARNING 53
WRITEBACK 54
CURSOR 56
PICK_LOCATION 57

Appendix A
Initial Function Instances by Category 58

Section RM3
Initial Function Instances

Whenever the PS 390 is booted, certain intrinsic functions are automatically instanced for use, and are called initial function instances. Initial function instances are of the form:

Function_instance_name

Unlike intrinsic functions, they are not preceded by F:. They provide access to host communication and to PS 390 interactive devices, as well as allowing for the display of messages on keyboard and control dial LEDs. Initial function instances are not used as templates to create uniquely named function instances. Instead, they are used in function networks by their own system-assigned name. They cannot be renamed by the user. For example,

```
SEND 'EXIT' TO <1> FLABEL12;
```

sends the string EXIT to the LED for function key F12.

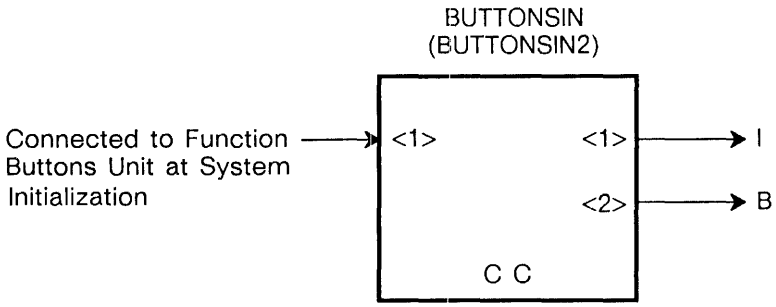
Initial function instances are listed alphabetically in this section. Not all PS 390 initial function instances are documented. Following the initial function instances, the initial structures CURSOR and PICK_LOCATION are documented. These establish the shape of the cursor as an "X" and the pick-sensitive location as the center of the cursor. Appendix A lists the initial function instances by category.

The following information, where relevant, is given for each Initial function instance and Initial structure:

- Name
- Type - Category
- Purpose
- Description of inputs and outputs
- Defaults
- Notes
- Associated Functions
- Examples

TYPE

Initial Function Instance — Data Input



PURPOSE

Detects activity from the function buttons unit on input <1>, which is connected to the system at initialization.

DESCRIPTION

INPUT

<1> — connected to function buttons

OUTPUTS

<1> — number of the button activated

<2> — TRUE = on, FALSE = off

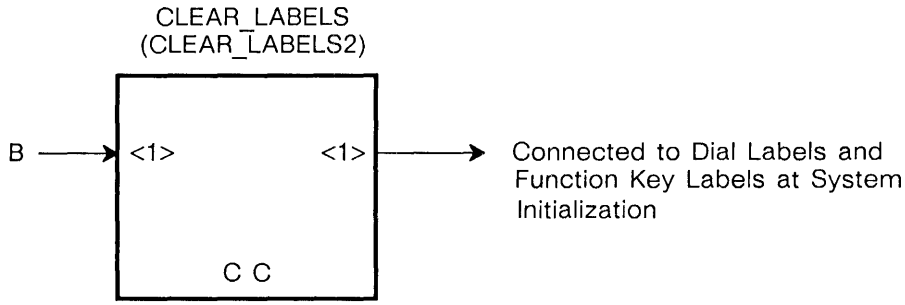
NOTE

Output occurs when one of the 32 buttons is pushed. The number of the pushed button appears at output <1>, and its light state (TRUE for on, FALSE for off) at output <2>.

CLEAR_LABELS

TYPE

Initial Function Instance — Data Output



PURPOSE

Clears the control dial LED labels and the function key LED labels. If input <1> is TRUE, the labels are cleared; otherwise, no action is taken.

DESCRIPTION

INPUT

<1> — TRUE = clear labels, FALSE = no action

OUTPUT

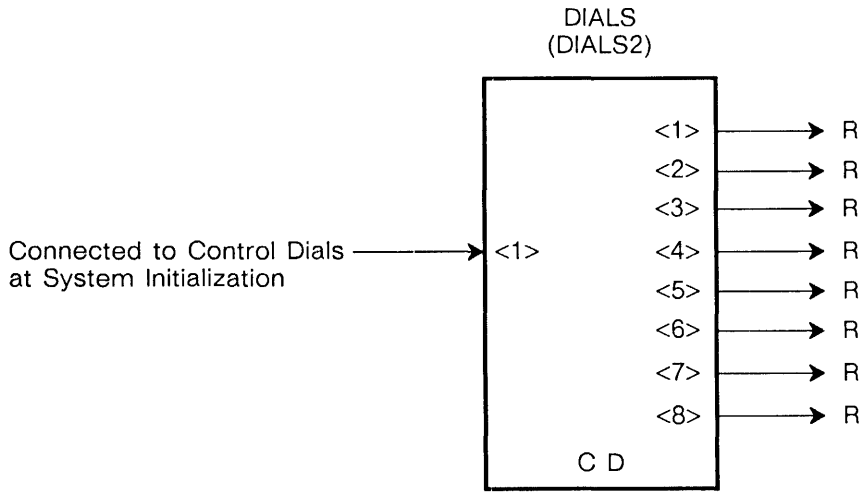
<1> — connected to dial and function key labels

NOTE

The INITIALIZE command sends a TRUE to this function instance, clearing all LED labels.

TYPE

Initial Function Instance — Data Input



PURPOSE

Produces eight real number outputs that correspond to inputs from control dials 1 though 8.

DESCRIPTION

INPUT

<1> — connected to control dials

OUTPUTS

<1> — real number

<2> — real number

<3> — real number

<4> — real number

OUTPUTS*(continued)*

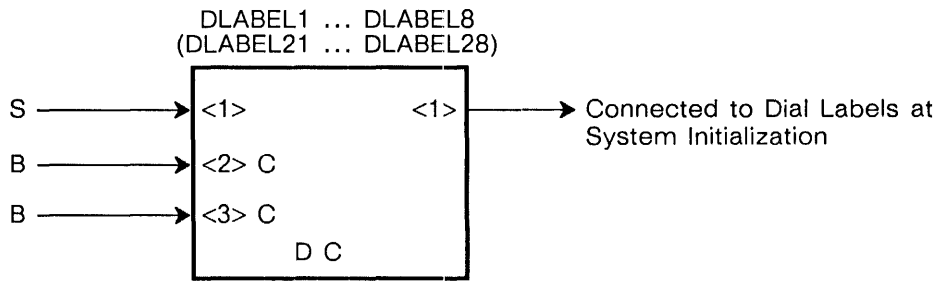
<5> — real number
<6> — real number
<7> — real number
<8> — real number

NOTES

1. The control dials are numbered from 1 through 4, left to right across the top row, and from 5 to 8, left to right across the bottom row.
2. The message from each control dial is converted to a real number value, which is the incremented value from the dial normalized to between –1.0 and +1.0. This value is sent out on the output (<1>...<8>) that corresponds to the number of the dial that sent the message.

TYPE

Initial Function Instance — Data Output



PURPOSE

Eight function instances are provided to separately label the LED indicators above each control dial. DLABEL1 is used to label the LED indicators associated with the first control dial (leftmost, top row); DLABEL2 is used to label the LED indicators associated with the second control dial (second from left, top row); and so on, through DLABEL8, which is used to label the LED indicators associated with the eighth dial (rightmost, bottom row).

DESCRIPTION

INPUTS

<1> — label message

<2> — blink/no blink (constant)

<3> — center/justify left (constant)

OUTPUT

<1> — connected to control dial labels

NOTES

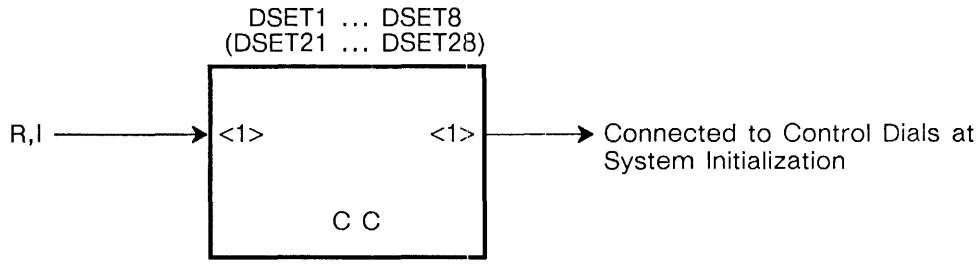
1. Input <1> accepts the character string (up to eight characters) to be displayed on the corresponding control dial LED indicators. The constant Boolean value on input <2> selects blink (TRUE) or no blink (FALSE) for the displayed characters. The constant Boolean value on input <3> controls whether the displayed message will be centered in the eight available locations (TRUE) or whether it will be justified left so that the first character is placed in the leftmost of the eight locations (FALSE).
2. If inputs <2> and <3> are not used, the message will not blink and will be centered.
3. Allowable characters for control dial labels are:

```
! " # $ % & ' ( ) * + - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ ] _
```

4. Lowercase letters are converted to uppercase letters. A space may also be specified.
5. Carriage return <CR> and line feed <LF> are not legal characters and cause a message that follows <CR> or <LF> to be partially lost or ignored.

TYPE

Initial Function Instance — Data Output



PURPOSE

Eight function instances are provided to set the operating parameters for the eight control dials. DSET1 is used to set parameters for the first control dial (leftmost, top row); DSET2 is used to set parameters for the second control dial (second from left, top row); and so on, through DSET8, which is used to set parameters for the eighth control dial (rightmost, bottom row).

DESCRIPTION

INPUT

<1> — real number = delta value, integer = sample rate

OUTPUT

<1> — connected to control dials

DEFAULTS

All control dials default to an enabled condition in relative mode (each value from a dial reflects the amount of change [delta] from the last output value). There is no absolute mode for the control dials.

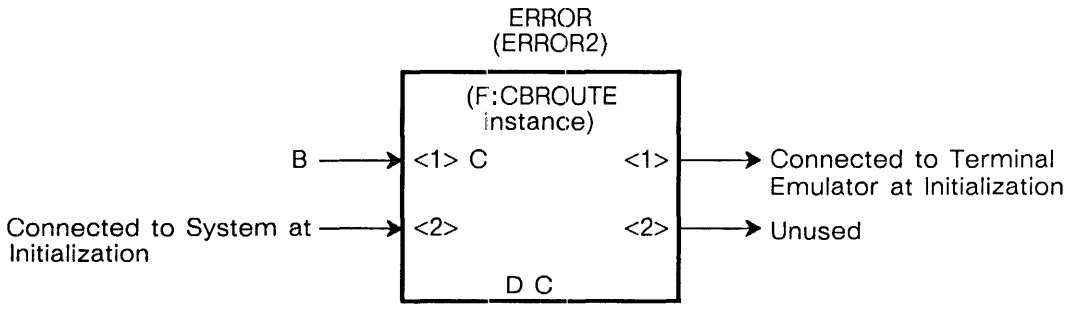
The default sample rate is 20 per second.

NOTES

1. Input <1> accepts real numbers or integers that set the delta value and sample rate. The default sample rate is 20 samples per second.
2. Real numbers set the delta value relative to one complete dial rotation. For example, if .25 were the real number input, the dial would have to be rotated 90 degrees (.25 x 360) before an output from the dial would be generated.
3. An integer is applied to input <1> to indicate the sample rate. Sample rate is specified in samples per second. For example, the integer 10 causes the dial to be sampled 10 times per second.
4. Output <1> is used to set the dial parameters as specified by the real number or integer on input <1>.

TYPE

Initial Function Instance — Miscellaneous



PURPOSE

Enables and disables the display of error messages.

DESCRIPTION

INPUTS

<1> — TRUE = enable, FALSE = disable (constant)

<2> — connected to system

OUTPUTS

<1> — connected to terminal emulator

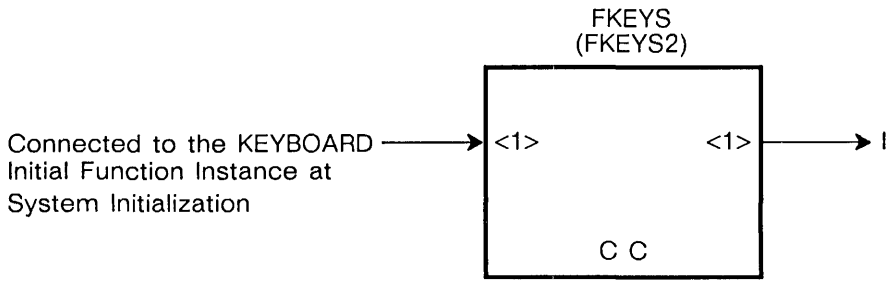
<2> — unused

NOTE

The INITIALIZE command automatically sends a TRUE to input <1> to enable the display of error messages.

TYPE

Initial Function Instance — Data Input



PURPOSE

Converts a character received from a keyboard function key to an integer code.

DESCRIPTION

INPUT

<1> — connected to KEYBOARD

OUTPUT

<1> — integer code (1-36)

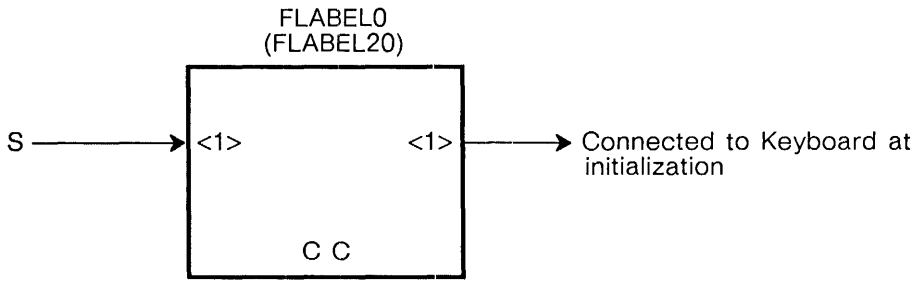
NOTE

Characters are converted as follows:

Integer Output	Corresponds To
1-12	Function keys F1-F12
13-24	Function keys F1-F12 with the shift key pressed.
25-36	Function keys F1-F12 with the control key pressed.

TYPE

Initial Function Instance — Data Output



PURPOSE

This initial function instance is similar to the FLABEL1 through FLABEL12 initial function instances in that it allows the user to specify characters to be displayed in the LED indicators above the function keys. However, unlike FLABEL1 through FLABEL12, which are used to separately specify the 8-character display above each function key, FLABEL0 allows a single character string (to a maximum of 96 characters) to be specified for display in the twelve 8-character displays. FLABEL0 treats the 96 LED displays as a single string of characters and spaces.

DESCRIPTION

INPUT

<1> — string for label

OUTPUT

<1> — connected to keyboard

NOTES

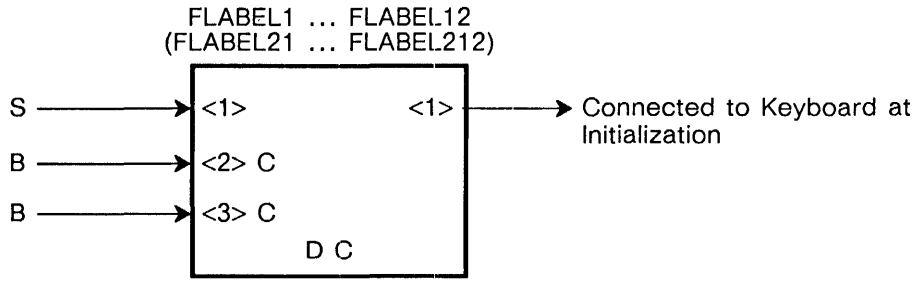
1. The string of characters on input <1> is displayed starting in the leftmost LED location (the first of the 8 LEDs over the first function key).
2. Allowable characters for the function key LED indicators follow:

   ```
   ! " # $ % & ´ ( ) * + - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
   A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ ] _
   ```

3. Lowercase letters are converted to uppercase letters for display. A space may also be specified.
4. Carriage return <CR> and line feed <LF> are not legal characters and cause a message that follows <CR> or <LF> to be partially lost or ignored.
5. This function is not valid if using a softlabels network.

TYPE

Initial Function Instance — Data Output



PURPOSE

Twelve initial function instances are provided to label the eight LED indicators above each of the twelve function keys. FLABEL1 is used to label the eight LED indicators associated with the first function key; FLABEL2 is used to label the eight LED indicators for the second function key; and so on, through FLABEL12, which is used to label the eight LED indicators for the twelfth function key.

DESCRIPTION

INPUTS

<1> — label message

<2> — blink/no blink (constant)

<3> — center/justify left (constant)

OUTPUT

<1> — string to function key LED

NOTES

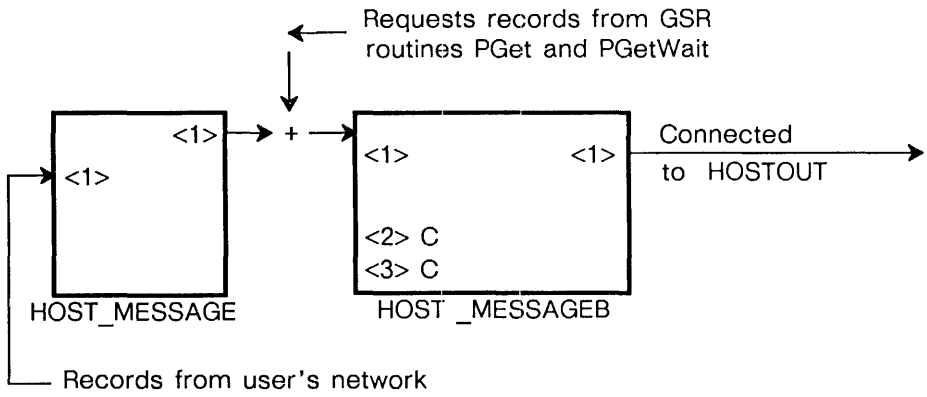
1. Input <1> accepts a character string (up to eight characters) to be displayed on the corresponding function key LED indicators. The constant Boolean value on input <2> selects blink (TRUE) or no blink (FALSE) for the displayed characters. The constant Boolean value on input <3> controls whether the displayed message will be centered in the eight available locations (TRUE) or whether it will be justified left so that the first character is placed in the leftmost of the eight locations (FALSE).
2. If inputs <2> and <3> are not used, the message will not blink and will be centered.
3. Allowable characters for Function Key labels are:

```
! " # $ % & ' ( ) * + - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ ] _
```

4. Lowercase letters are converted to uppercase letters for display. A space may also be specified.
5. Carriage return <CR> and line feed <LF> are not legal characters and cause a message that follows <CR> or <LF> to be partially lost or ignored.
6. The FLABEL1 through FLABEL12 function instances are used to separately program the 8-character LED displays associated with a particular function key. If the entire set of 96 LEDs (12 function keys x 8 characters per function key) is to be programmed as a single message, then FLABEL0 must be used.

TYPE

Initial Function Instance — Miscellaneous



PURPOSE

HOST_MESSAGEB is an initial function instance of the function F:HOLDMESSAGE. It supports the PGetW and PGet FORTRAN routines and the PGetWait and PGet Pascal and UNIX routines. All messages are sent from the PS 390 to the host via this function when using the GSRs.

HOST_MESSAGE is an initial function instance of the function F:NOP. It is recommended that the user always send PS 390 output destined for the host to HOST_MESSAGE rather than HOST_MESSAGEB, since the name of the latter may change with a future release of runtime software.

The routines PGet and PGetW specifically interrogate the initial function instance HOST_MESSAGEB for input back to the host.

The routine PGet is used to "poll" the PS 390 for data. If a message exists on the queue of HOST_MESSAGEB, then that message is removed from the queue and is returned by PGet. If no message was present in the input of HOST_MESSAGEB, then the special "no-messages" message as defined by input <3> of HOST_MESSAGEB is returned.

The routine PGetW is similar in functionality to PGet with one important difference. PGetW will not return to the caller until a message has been received from the PS 390. If no messages are present on the input of HOST_MESSAGEB, then the caller of PGetW (Get message and wait for completion) will wait until a message is sent to input <1> of HOST_MESSAGEB.

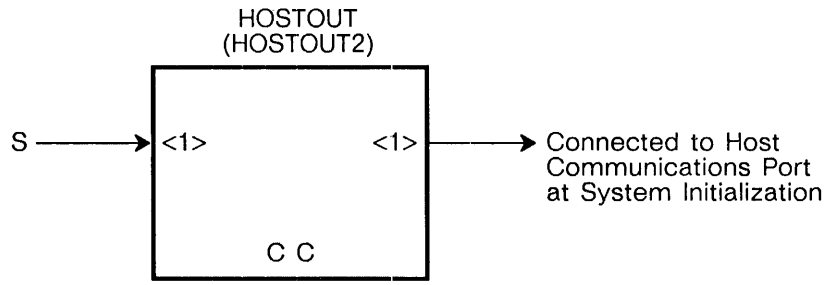
Messages received from the PS 390 via PGet and PGetW may need to be trimmed of the trailing character(s) as defined by inputs <2> and <3> of HOST_MESSAGEB if either of them is changed from the default value of carriage return (Character 13). The GSR will remove a single trailing carriage return from the message. Thus, if a poll operation is requested and no messages are present, the GSR returns a zero-length message to the caller indicating that no messages were present because the default "no-message" message on input <3> of HOST_MESSAGEB is a carriage return. Similarly, calls to PGetW return the proper length. However, if the user chooses to change the HOST_MESSAGEB inputs <2> and <3>, then the user must compensate for any side effects so produced when calling PGet or PGetW.

EXAMPLE

Refer to Helpful Hint 14 in Section *TT2*.

TYPE

Initial Function Instance — Data Output



PURPOSE

Accepts a string on input <1> and outputs the string for communication to the host. At initialization, the input to HOSTOUT is connected to a network used for host communications.

DESCRIPTION

INPUT

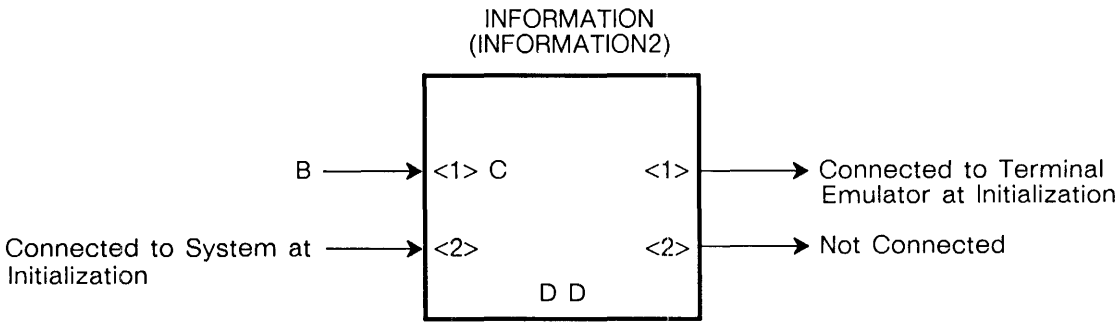
<1> — string to be sent to host

OUTPUT

<1> — connected to host communications port

TYPE

Initial Function Instance — Miscellaneous



PURPOSE

Enables and disables the display of information messages.

DESCRIPTION

INPUTS

<1> — TRUE = enable, FALSE = disable

<2> — connected to system

OUTPUTS

<1> — connected to terminal emulator

<2> — not used

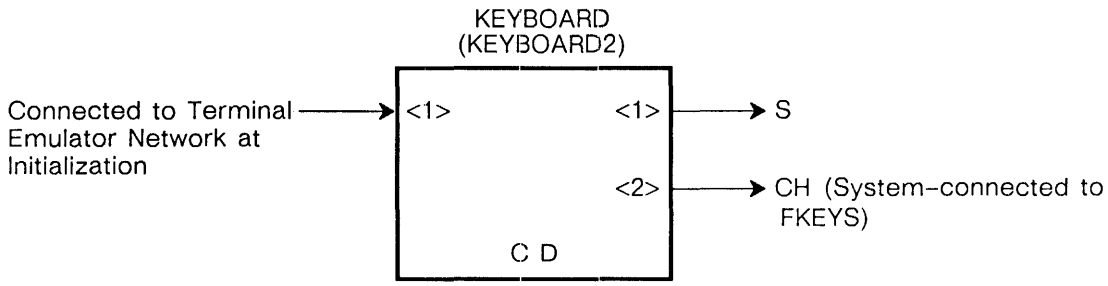
NOTE

The INITIALIZE command automatically sends a TRUE to input <1> to enable the display of information messages.

KEYBOARD

TYPE

Initial Function Instance — Data Input



PURPOSE

Connected at initialization to accept an ASCII character string from the keyboard.

DESCRIPTION

INPUT

<1> — connected to terminal emulator network

OUTPUTS

<1> — characters not preceded by CTRL/V

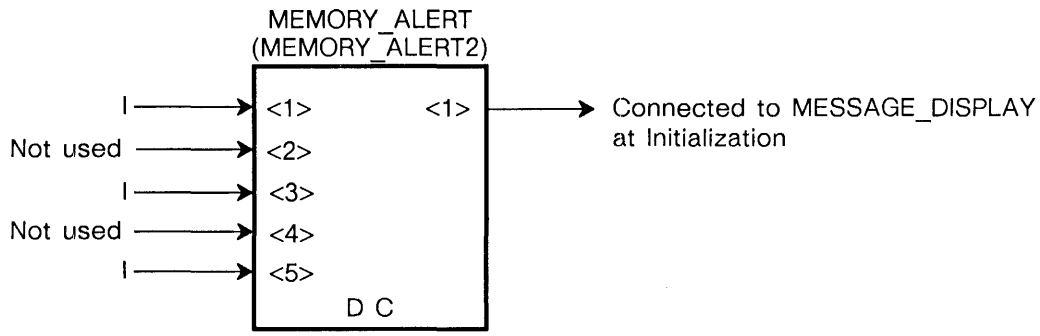
<2> — character preceded by CTRL/V

NOTE

Input characters are checked for a preceding CTRL/V character. If a character is preceded by a CTRL/V, the CTRL/V is removed by the function and the associated character is output on <2>, which is system-connected to the input of FKEYS. Characters that are not preceded by a CTRL/V are output on <1>.

TYPE

Initial Function Instance – Miscellaneous



PURPOSE

Generates a message and a bell alarm when system memory usage reaches a specified threshold.

DESCRIPTION

INPUTS

<1> — memory threshold percentage for reporting

<2> — unused

<3> — sampling interval

<4> — unused

<5> — integer specifying memory threshold in bytes for vector-list creation

OUTPUT

<1> — connected to MESSAGE_DISPLAY

DEFAULTS

The threshold specified on input <1> is set at 75% unless changed by the user. The sampling interval is set at 10 seconds unless changed. Input <5> defaults to 0.

NOTES

1. The number of bytes specified on input <5> is the minimum number of bytes that must be available in memory for the system to create vector list. Once this threshold has been reached, a vector list will be only partially created, or not created at all. When this occurs, the error message "E 105 *** cannot complete operation due to insufficient memory" is issued. This applies to vector lists, characters, labels, polynomials, b-splines, patterned vector lists, and polygons.
2. Memory status is sampled at 10–second intervals. The message displayed is of the form:

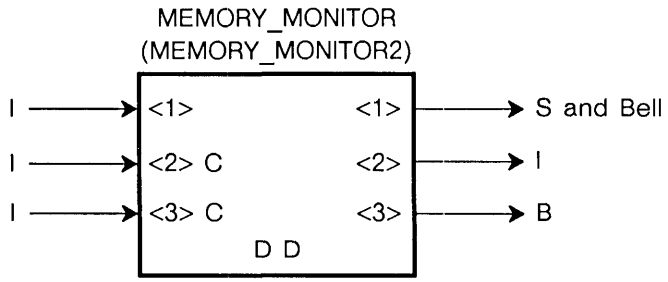
```
MASS MEMORY nn PERCENT FILLED.
```

3. If the amount of memory used falls below the threshold, the message is removed.
4. Output <1> is connected to MESSAGE_DISPLAY at initialization.
5. If the user wishes to change the percentage that generates the alarm, another value must be sent to input <1>. If the user wishes to specify a sampling interval other than 10 seconds, another value must be sent to input <3>. The value is an integer that specifies the number of seconds to wait before rechecking memory.

MEMORY_MONITOR

TYPE

Initial Function Instance — Miscellaneous



PURPOSE

Notifies the user of the number of bytes that are available for use out of a maximum number of bytes available at system initialization and of the elapsed time since initialization.

DESCRIPTION

INPUTS

<1> — memory threshold percentage

<2> — delta value (constant)

<3> — sampling interval (constant)

OUTPUTS

<1> — message string and bell

<2> — percentage full

<3> — TRUE if threshold is exceeded, not sent if otherwise

DEFAULTS

The threshold is set at 75%, the delta value is set at 0, and the sampling rate is set at 10 seconds unless changed by the user.

NOTES

1. None of the outputs from this function instance is connected upon system initialization. The user must connect output <1> of MEMORY_MONITOR input <1> of MESSAGE_DISPLAY. This causes the message to be displayed in the message display area of the screen and a bell to be sent to the keyboard. The message displayed is of the form:

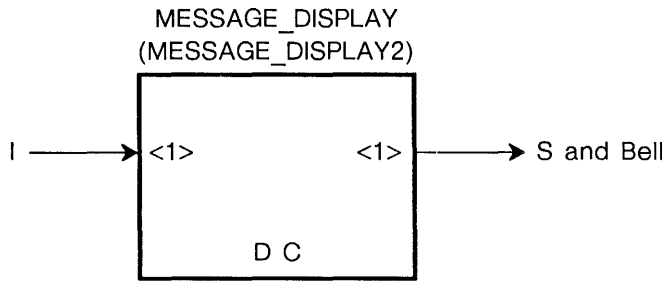
```
nnnnnn bytes free out of nnnnnnnn bytes maximum at dd hh:mm:ss
```

2. Output <2> is an integer representing the percentage of memory filled. Unless a change on input <2> (since the last report) is equal to or greater than the previous value on input <2>, no report is given.
3. Output <3> is a Boolean value that is output as a TRUE if the threshold indicated on input <1> is crossed.

MESSAGE_DISPLAY

TYPE

Initial Function Instance — Miscellaneous



PURPOSE

Displays error messages and informational messages in the MESSAGE_DISPLAY area of the PS 390 display. At initialization, input <1> is connected by the system to output <1> of MEMORY_ALERT and error-detection functions. Output <1> is connected to input <1> of FLABEL0 so that bell messages can be sent to the keyboard.

DESCRIPTION

INPUT

<1> — connected to MEMORY_ALERT and error-detection functions

OUTPUT

<1> — string and bell connected to FLABEL0

NOTE

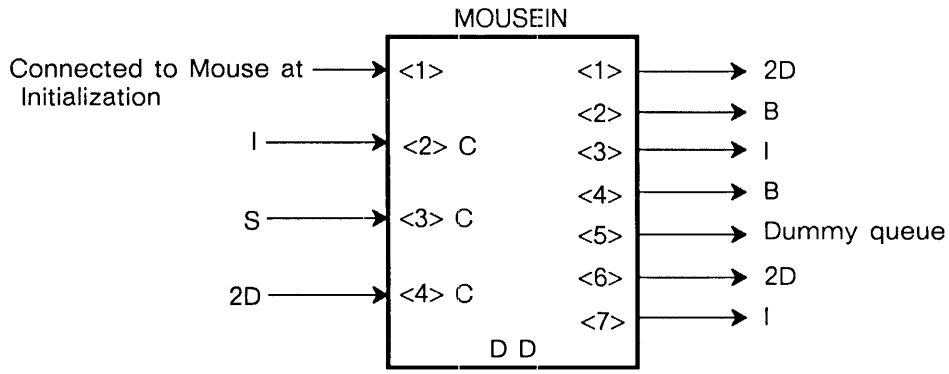
Each string received is treated as a complete message. Incoming characters are displayed at position 1 and replace the previous message.

EXAMPLE

Refer to Helpful Hint 14 in Section *TT2*.

TYPE

Initial Function Instance — Data Input



PURPOSE

Connected at system initialization to accept data from the mouse on input<1>. MOUSEIN is similar to TABLETIN.

DESCRIPTION

INPUTS

<1> — trigger (connected to mouse at firmware load)

<2> — integer (counts full scale of work area)

<3> — string (define active outputs)

<4> — 2D vector (X,Y cursor position)

OUTPUTS

<1> — normalized 2D vectors

<2> — mouse switch open/closed (TRUE=closed)

<3> — raw switch byte (bit encoded) (sum of the buttons that are pressed)

OUTPUTS *(continued)*

<4> — tip-switch transitioned to state
<5> — dummy output queue for TABLETIN compatibility
<6> — 2D vector on true edge detect
<7> — switch number translated to range 0–3

NOTES

1. The mouse protocol is as follows:

Bit No.	7	6	5	4	3	2	1	0
Byte 1	1	0	0	0	0	L	M	R
Byte 2	X	X	X	X	X	X	X	X
Byte 3	Y	Y	Y	Y	Y	Y	Y	Y
Byte 4	X	X	X	X	X	X	X	X
Byte 5	Y	Y	Y	Y	Y	Y	Y	Y

   L,M,R refer to the Left, Middle and Right buttons. There are two relative change samples of X and Y in two's complement format. To determine the actual change in position, the two X values must be added together and the two Y values must be added together.

2. When the MOUSEIN function is active and output <3> is enabled, the possible outputs are 0–7. If no buttons are pressed, the integer 0 is sent on output <3>. The following integer values correspond to the buttons pressed:

```
    R -> 1
    M -> 2
  M+R -> 3
    L -> 4
  L+R -> 5
  L+M -> 6
L+M+R -> 7
```

3. When the MOUSEIN function is active and output <7> is enabled, the possible outputs are 0–3. If no buttons are pressed the integer 0 is sent on output <7>. The following integer values correspond to the buttons pressed:

```
    R -> 1
    M -> 2
  M+R -> 2
    L -> 3
  L+R -> 3
  L+M -> 3
L+M+R -> 3
```

4. Input <1> accepts input from the mouse. This input is connected by the system at firmware loading. When the function activates, it may have part of a report, many reports, or any combination.

5. Input <2> counts full scale. Theoretically, this is the dimension of the work area in inches multiplied by the resolution in counts per inch (200).

6. Input <3> defines which outputs are to be used and defines cursor update enable/disable. An 8-character string is expected on input <3>, where each position in the string represents an output queue. A string of less than seven characters defaults to all T. Following are two examples:

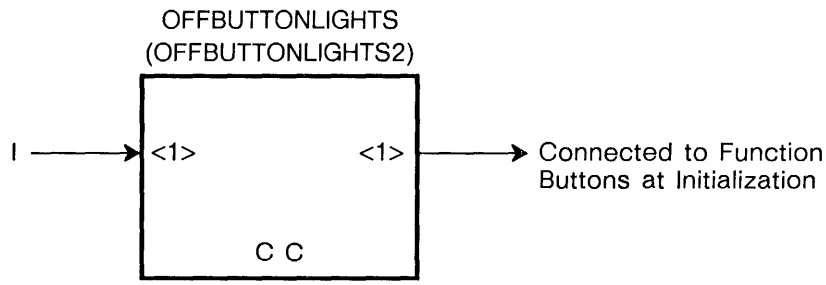
   `'TFFFFFFT'` (queue 1 enabled—first T; all others disabled; cursor will be updated—last T)

   `'T'` (all queues enabled; cursor updating enabled)

OFFBUTTONLIGHTS

TYPE

Initial Function Instance — Data Output



PURPOSE

Turns off lighted buttons on the function buttons unit.

DESCRIPTION

INPUT

<1> — integer (1–32) indicating the button number

OUTPUT

<1> — connected to function buttons

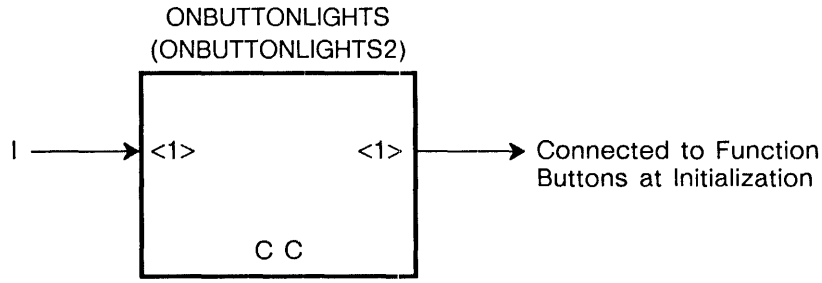
NOTES

1. Each button may be turned off independently or all buttons may be turned off by a single message. A zero (0) or any out–of–range integer at input <1> turns off all button lights. An integer from 1 to 32 at input <1> turns off the corresponding button light.
2. Function buttons are arranged in one row of four, four rows of six, and another row of four. They are numbered from left to right starting from the top row. The top row is numbered 1 through 4; the second row 5 through 10, and so on until the last row, 29 through 32.

ONBUTTONLIGHTS

TYPE

Initial Function Instance — Data Output



PURPOSE

Turns on lighted buttons on the function buttons unit.

DESCRIPTION

INPUT

<1> — integer (1–32) indicating the button number

OUTPUT

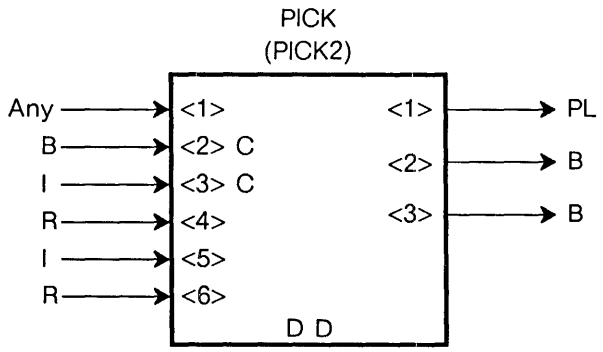
<1> — connected to function buttons

NOTES

1. Each button may be turned on independently or all buttons may be turned on by a single message. A zero (0) or any out-of-range integer at input <1> turns on all button lights. An integer from 1 to 32 at input <1> turns on the corresponding button light.
2. Function buttons are arranged in one row of four, four rows of six, and another row of four. They are numbered from left to right starting from the top row. The top row is numbered 1 through 4; the second row 5 through 10, and so on until the last row, 29 through 32.

TYPE

Initial Function Instance — Data Input



PURPOSE

Interfaces with the hardware picking circuitry. Any message on input <1> arms the PICK function. Once PICK is enabled, when a pick occurs, the pick list associated with the picked data is sent out on output <1> and a Boolean FALSE is sent out on output <2>. Typically, this Boolean value is used to disable picking of a set of objects by connecting it to a SET PICKING ON/OFF node in a display structure.

DESCRIPTION

INPUTS

<1> — function trigger

<2> — TRUE = coordinate, FALSE = index (constant)

<3> — timeout duration (constant)

<4> — defines pick window half size for the ACP pass of the pick

<5> — retry count

<6> — half-size increment to be added to window half-size on each retry

OUTPUTS

<1> — pick list

<2> — FALSE = pick enabled

<3> — FALSE = timeout elapsed

NOTES

1. Input <2> selects the kind of pick list that will be output on output <1>. A FALSE on input <2> indicates that the output pick list will be the pick identifier and an index into the vector list or the character string. (The index into the vector list identifies its position in the list; e.g., vector 3 is the third vector in a vector list. The index into a character string identifies the picked character by its position in the string; character 5 is the fifth character in a string.)
2. A TRUE on input <2> indicates that the output pick list will include, in addition to the pick identifier and the index, the picked coordinates and the dimension of the picked vector. If the vector is part of a polynomial curve, its parameter value, t, is supplied instead of the index.
3. Coordinate picking on a character string returns an index into the string, not its picked coordinates.
4. Coordinate picking cannot be performed on a vector over 500 [LENGTH] units long.
5. The pick list on output <1> is typically connected to an instance of F:PICKINFO to convert the pick list to a locally useful format. If the pick list is to be printed out, output <1> may be connected to F:PRINT to convert the pick list code to printable characters.
6. When several vectors are picked, the first vector drawn by the line generator is reported as picked. For example, if three vectors in a single vector list were picked simultaneously (at a point of intersection), the first vector listed in the object definition would be reported as picked.

7. The integer on input <3> specifies a pick timeout period in refresh frames. This pick timeout period allows the user to determine whether a pick has occurred within the specified amount of time. Timing starts when the PICK function is armed with a message on active input <1>. Allowable integers for input <3> are from 4 through 60.

8. If input <3> is not used, all picks will be reported once the function is armed because no timeout duration has been specified.

9. Typically, the FALSE at output <3> would be used to turn off picking in a display structure (at a SET PICKING ON/OFF node) or to send a "NO PICK" message (probably via F:SYNC(2)) back to the host.

10. The user has three means of canceling an existing pick timeout duration:

    a. Send an INITIALIZE command. This will remove the PICK function and replace it with a new instance of the PICK function.

    b. Send a non-integer (and ignore the "Bad message" error).

    c. Send an integer less than 4 or greater than 60 to input <3> (and ignore the "Bad message" error).

11. Input <4> is a real number between 0 and 1 that defines the pick window half-size for the ACP pass of the pick. This is different from the size set by the SET PICKING LOCATION operation node. The line generator or the frame buffer uses the operation node to determine if a pick has occurred; whereas the ACP uses input <4> to do the actual pick pass on the data. Default is 6.8359E–3.

12. Input <5> is an integer specifying pick pass retries. Since it is possible that the ACP will not find the picked data during a pick pass, input <5> indicates the number of times to add the window increment on input <6> and try another pick pass. The default is 4.

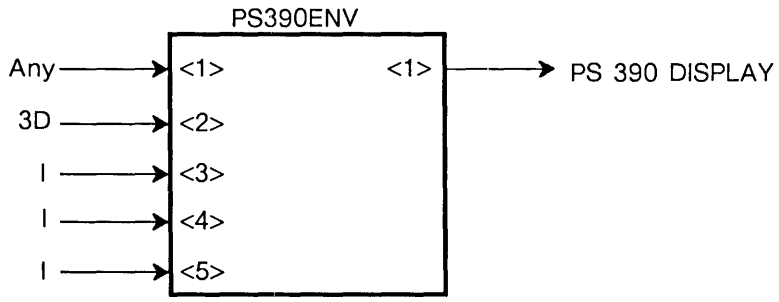
13. Input <6> is a real number between 0 and 1 which specifies the amount to increase the pick window half-size on each retry of the pick pass. Default is 6.8359E–3.

EXAMPLE

If a 10 is sent to constant input <3>, then the PICK function is armed with a message on input <1>. The function waits 10 refresh frames from the time the input <1> message is received before checking to see if a pick has occurred. If a pick has occurred within that period, the function outputs the appropriate pick list. If a pick has not occurred, the function outputs a FALSE on output <3>. In either case, the PICK function is disarmed and must be rearmed via input <1> before further picking can be reported.

TYPE

Initial Function Instance — Miscellaneous



PURPOSE

This initial function instance sets up display background color, and selects cursor and cursor color.

DESCRIPTION

INPUTS

<1> — trigger which accepts any data type to cause the function to run.

<2> — constant which accepts a 3D vector (hue, saturation and intensity) to specify background color.

<3> — constant which accepts an integer in the range [0,7] to specify the system-defined refresh cursor color, where:

```
0 = black
1 = blue
2 = green
3 = cyan
4 = red
5 = magenta
6 = yellow
7 = white (default)
```

INPUTS *(continued)*

<4> — constant which accepts an integer to select the cursor

```
0 = update rate cursor (default)
1 = system=defined refresh cursor
```

<5> — accepts an integer to specify the video timing format, which is output from the video connection on the back of the PS 390 control unit

```
0 = 1024 x 864 non-interlaced (default
    required by the PS 390 display)
2 = 1024 x 864 interlaced
3 = 640 x 484 interlaced (RS-170)
```

OUTPUT

<1> — connected to the PS 390 display

DEFAULTS

The default background color on input <2> is 0,0,0 (black). Saturation and Intensity must be in the range of [0,1], otherwise an error message will be generated. Hue is in the range of [0,360]. For any value specified outside this range, multiples of 360 are added or subtracted to bring it into this range.

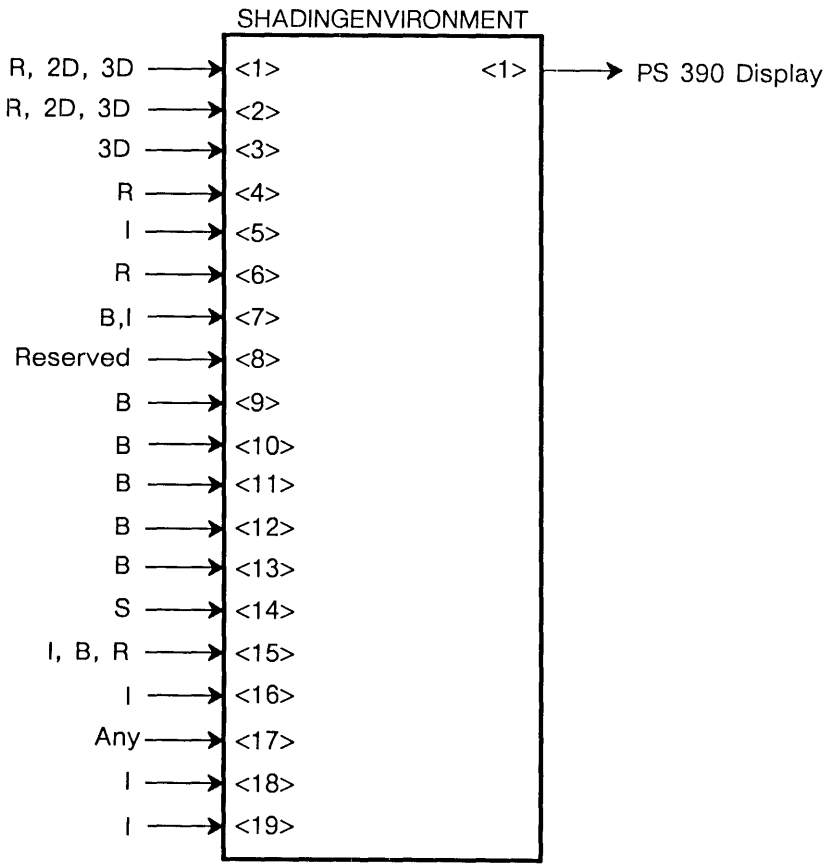
On input <3> any value outside [0,7] generates an error.

NOTE

When specifying the system-defined refresh rate cursor, you should leave the initial viewports HVP1$ and GVP0$ unchanged in order to have the (hardware) cursor work with picking.

TYPE

Initial Function Instance — Miscellaneous



PURPOSE

This function allows you to control various aspects of shaded renderings displayed in the static viewport.

DESCRIPTION

INPUTS

<1> — ambient color

<2> — background color

INPUTS *(continued)*

<3> — static viewport
<4> — exposure
<5> — edge-smoothing (antialiasing) control
<6> — depth cueing
<7> — screen wash
<8> — reserved
<9> — overlay/refresh control
<10> — color by vertex control
<11> — opaque (transparency) control
<12> — specular highlights control
<13> — special color blending for spheres control
<14> — update attribute table
<15> — line Z-value control (polygon edge enhancement)
<16> — resolve rendering visibility
<17> — restore gamma-corrected system look-up table
<18> — vertex normal control
<19> — stereo rendering

OUTPUT

<1> — connected to the PS 390 display

NOTES

1. Ambient color: input <1> accepts a real number as hue, a 2D vector as hue and saturation, and a 3D vector as hue, saturation, and intensity, to specify the ambient color. The ambient color is combined with the result obtained from the light sources to determine the color of ambient light. The default ambient color is white, with a default intensity of 0.25.

2. Background color: input <2> accepts a real number as hue, a 2D vector as hue and saturation, and/or a 3D vector as hue, saturation, and intensity to specify the background color. Refer to the COLOR parameter of the ATTRIBUTES command for the meaning of the values. The current static viewport will be colored with the background color prior to any shaded rendering done in the refresh mode (refer to input <9>). The default background color is black (0,0,0).

3. Static viewport: input <3> accepts a 3D vector which specifies physical pixel locations for the viewport where shaded renderings are displayed. Static viewports are always square, the lower left corner being given by the X and Y coordinates of the vector, and its size given by the Z coordinate, such that the upper right corner is at (X+Z,Y+Z). Values are rounded to the nearest pixel. The default viewport is V3D(80,0,863).

   The viewport can be used for rendering multiple images side by side on the raster display. For example, **send V3D(0,-80,1023)** would be a valid command to specify the largest recommended value for the static viewport. This viewport encompasses the entire displayable screen as well as the undisplayable area in Y that is in excess of 863. Images in this viewport are clipped to the physical raster for which $0 \leq X < 1024$ and $0 \leq Y < 864$.

4. Exposure: input <4> accepts a real number as the exposure, controlling the overall brightness of the picture. The exposure is like that on a camera. If a picture is taken of an object with a very bright specular highlight, it may be so bright that the rest of the object is darkened. If three light sources exist, the object would be about three times brighter, making the object too bright. The exposure can be brought down to control this.

   The exposure is multiplied by the intensity at each pixel and the result clipped to the maximum intensity. This enables the overall brightness of a rendering to be increased without causing bright spots to exceed maximum intensity (instead forming "plateaus" of maximum intensity). Recommended exposure values may vary between 0.3 and 3.0. The default exposure is 1.

5. Edge smoothing (antialiasing) control: input <5> accepts an integer which allows users to choose between having a relatively fast rendering with jagged edges along the edges of polygons or having a slower rendering with smoother edges. Antialiasing is accomplished by taking 16 samples per pixel instead of only one. You are given the choice of having no edge smoothing at all, smoothing along the edges only, or sampling 16 times within every pixel for every polygon. The default value in this input is 0.

   Sending Fix(0) to this input produces no smooth edges, and produces the fastest renderings. Polygons are rendered with one sample per pixel.

   Sending Fix(1) produces smooth edges, but may not correctly resolve visibility between surfaces that are extremely close in z-values or that are interpenetrating. The 16 samples are taken only where the edges of the polygon touch a pixel. The interior of the polygons are still rendered with one sample per pixel. This has a speed intermediate between a Fix(0) and a Fix(2).

   Sending Fix(2) to this input produces full antialiasing. This method produces the slowest renderings, but it produces full visibility resolution. 16 samples are taken for every pixel in every polygon.

6. Depth cueing: input <6> accepts a real number in the range of 0 to 1 to control depth cueing in the shaded image (1 specifying no depth cueing and 0 specifying maximum depth cueing). As perceived depth from the viewer increases, the colors are mixed with the ambient light color. Thus, if a 3D vector(0,0,0) (black) is sent to the ambient input <1> and if a 0 is sent to input <6>, the objects will be rendered with a ramp ending in black at the back clipping plane. A 1 sent to input <6> will turn off depth cueing. The default is 0.2 giving a fairly large depth cueing effect.

7. Screen wash: input <7> accepts a Boolean value or an integer, and produces an immediate visual effect. Sending a TRUE to this input clears the entire screen to static and causes a screen wash with the current static background color. Sending FALSE to this input clears the currently specified static viewport and causes the viewport to be filled with the current static background color.

Sending Fix(0) to input <7> has the same effect as sending TRUE.

Sending Fix(1) to input <7> has the same effect as sending FALSE.

Sending Fix(2) to input <7> clears the entire screen to dynamic and causes a screen wash with the current dynamic background color. This may be done to clear a shaded image before displaying a new dynamic image.

Sending Fix(3) to input <7>, clears the currently specified static viewport with the current dynamic background color.

8. Reserved

9. Clear/Overlay Control: input <9> accepts a Boolean value which determines whether the screen is to be cleared with the current background color before the rendering is performed. Sending a TRUE to this input causes the current object to be rendered on top of the image currently being displayed in the static viewport. Sending FALSE causes the static viewport to be washed clean with the current background color before an object is rendered. The default is FALSE.

10. Color by Vertex Control: input <10> accepts a Boolean value which turns off or on the use of vertex colors. Color by vertex is accomplished by defining a color for each vertex in the polygon. A TRUE sent to this input enables the colors to be defined at each vertex. A FALSE sent to this input enables the colors specified in the ATTRIBUTES command. The default value for this input is FALSE.

11. Opaque (Transparency) Control: input <11> accepts a Boolean value which enables or disables the transparency assigned to the polygon with the OPAQUE clause of the ATTRIBUTE command. Transparent polygons are created by modifying the ATTRIBUTE command as follows:

```
Name := ATTRIBUTE    [Color h[,s[,i] ] ] [OPAQUE t]
                     [Diffuse d] [Specular s];
```

where t refers to a value between 0 and 1, with 1 being a fully opaque polygon and 0 being a fully transparent polygon. When t=0, the object is completely invisible. The default value for this input is FALSE (fully opaque).

12. Specular Highlight Control: input <12> accepts a Boolean value which allows you to turn specular highlights on and off. Flat, Gouraud, and Phong shading use a shading equation that can process multiple light sources and calculate specular highlights. The default value is TRUE, which means that specular highlights are turned on.

13. Special Color Blending for Spheres: input <13> accepts a Boolean value which turns off or on the color blending used for correct spherical rendering. Sending a TRUE turns on special color blending. Sending a FALSE turns off special color blending. The default is FALSE.

14. Update Attribute Table: input <14> accepts a string which is the name of a 3D tabulated vector list to update the attribute table that specifies color, radii, diffuse, and specular highlights for spheres and lines. The attribute table has 0 to 127 entries with six table components for each entry. The attribute table can be updated by encoding the table entries into a named PS 390 vector list and then sending the name of the vector list to input <14>. The six table components are encoded into two consecutive 3D tabulated vectors of the vector list.

    The table has the following components: hue, saturation, intensity, radius, diffuse, and specular. Hue is a real number in the range 0 to 360. Saturation and intensity are real numbers in the range 0 to 1. Radius is a real number greater than 0. Diffuse is a real number in the range 0 to 1. Specular is an integer in the range 0 to 255.

    The table is initialized as follows:

INDEX	Hue	Sat	Intensity	Radius	Diffuse	Specular
0	0	0	0.5	1.8	0.7	4 (Gray)
1	0	0	1	1.2	0.7	4 (White)
2	120	1	1	1.35	0.7	4 (Red)
3	240	1	1	1.8	0.7	4 (Green)
4	0	1	1	1.8	0.7	4 (Blue)
5	180	1	1	1.7	0.7	4 (Yellow)
6	0	0	0.7	1.8	0.7	4 (Gray)
7	300	1	1	2.15	0.7	4 (Cyan)
8	60	1	1	1.8	0.7	4 (Magenta)
9	0	0	0	1.8	0.7	4 (Black)
10-127						(Color Wheel)

Spheres use all six of these components. Lines use only the hue, saturation, and intensity components.

The t field of each 3D tabulated vector is used as an index into this table.

15. Raster Lines Z-value Control: input <15> accepts a Boolean value, or a real number in the range 0–1, or an integer in the range 0–2. A real value sent to this input adds an offset to the Z-values of lines causing them to be displayed in front of other objects with the same Z-value.

    Sending a Boolean value to this input allows you to toggle the display of polygon edges on and off. A TRUE causes lines to be drawn along polygon borders, thus enhancing the edges, and temporarily turns on full antialiasing. A FALSE causes polygons to be rendered normally without edge enhancement.

    Sending Fix(0) to <15> causes polygons to be rendered without enhanced edges (edges shown on the rendered image). This is the same as sending a FALSE.

Sending Fix(1) causes polygon edges to be enhanced, and causes all edges including those marked as soft to be displayed. This is the same as sending a TRUE.

Sending Fix(2) causes polygon edges to be enhanced, but only those edges marked as hard edges are displayed.

16. Resolve rendering visibility: input <16> accepts an integer value of 1 or 0 to choose between one of two possible algorithms for resolving visibility in a rendering.

    Sending Fix(0) causes a scan-line Z-buffer algorithm to be applied. This algorithm is used in rendering solids; it causes all backfacing polygons to remain undisplayed.

    Sending Fix(1) causes the painters algorithm to be applied to the rendering. This algorithm renders an image by filling (painting) each polygon.

17. Restore gamma-corrected system look-up table: any value sent to this input restores the gamma-corrected system look-up table which is the table responsible for producing antialiased lines of good line quality. Sending a value to this input has an immediate visual effect.

18. Vertex normal control: input <18> accepts an integer value in the range [0..2]. Values sent to this input affect vertex normals.

    Sending Fix(0) causes vertex normals to remain unchanged from their original definition. This is the default.

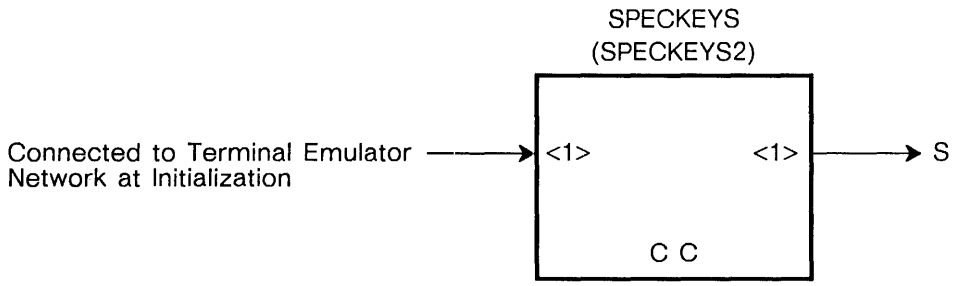
    Sending Fix(1) inverts vertex normals that are backwards and that are on backfacing polygons to make the polygons appear forward. This is useful for the user who knows the desired direction for normals to point, but who does not necessarily specify polygon vertices in a consistently clockwise fashion. This is applicable to surface renderings only. The AND specifier of the ATTRIBUTES command should not be used when using this input to reverse normals.

    Sending Fix(2) inverts vertex normals that are backwards and are on polygons that are front-facing to make the polygons appear forward. This is useful for performing mirrored modeling operations; i.e., using a -Y scale factor to produce an image mirrored about the XZ plane. Again, this is applicable only to surface renderings.

19. Stereo Rendering: Sending Fix(1024) to input <19> causes renderings to be produced on the entire display, including the (usually) missing 160 scan lines at the bottom of the screen. This input is used for rendering solid polygons, spheres, and raster lines in 3 dimensional stereo (using the Tektronix LCD screen).

TYPE

Initial Function Instance — Data Input



PURPOSE

Connected at initialization to accept an ASCII character string from the keyboard. Input characters are checked for a preceding CTRL/V character. If a character is preceded by a CTRL/V, SPECKEYS removes the CTRL/V and outputs the associated character on output <1>. (Characters not preceded by a CTRL/V appear at the output of KEYBOARD instead.)

DESCRIPTION

INPUT

<1> — connected to terminal emulator

OUTPUT

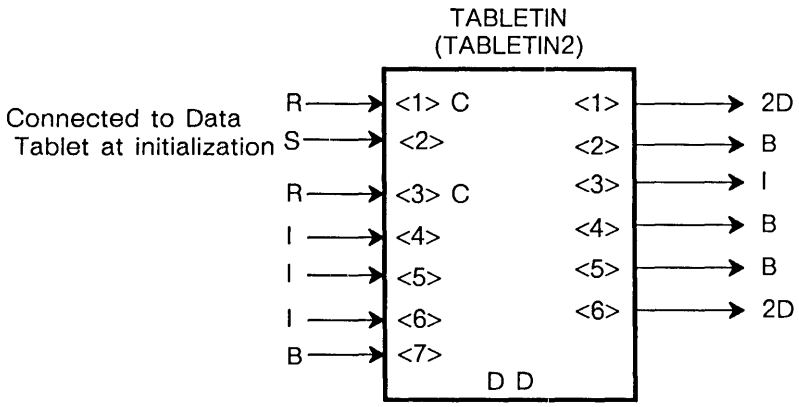
<1> — string

NOTE

Note that neither SPECKEYS nor KEYBOARD outputs function key values. The Initial Function Instance FKEYS supplies these values.

TYPE

Initial Function Instance — Data Input



PURPOSE

Connected at system initialization to accept data from the data tablet on input <2>. This data includes 2D vectors, an indication of the open/closed condition of the stylus tipswitch (or 4-button cursor), and an indication of the switch number for systems using a 4-button cursor instead of a stylus.

DESCRIPTION

INPUTS

<1> — delta X,Y (constant)

<2> — string (system connected to tablet)

<3> — tablet size (constant)

<4> — wait time

<5> — X origin of surface

<6> — Y origin of surface

<7> — TRUE = cursor ON, FALSE = cursor OFF if puck is out of proximity

OUTPUTS

<1> — X,Y coordinate (position/line)
<2> — TRUE = switch closed, FALSE = switch open
<3> — switch number
<4> — tipswitch transition
<5> — range transition
<6> — X,Y when switch closed

DEFAULTS

The default delta X,Y on input <1> is .002. The default tablet size on input <3> is 2200. The default wait time on input <4> is 8 centiseconds.

NOTES

1. Input <1> accepts a real number that specifies the minimum change in X or Y required on input <2> before output <1> is sent. The default value is .002.
2. Input <3> accepts an integer that specifies the number of points full-scale for the data tablet being used. The default value is 2200, corresponding to the standard 11-inch x 11-inch data tablet.
3. Input <4> is a wait time for the data tablet in centiseconds; a FALSE is sent on output <5> if the tablet stops sending data for longer than this duration. The default value is 8. It should never be necessary to SEND to this input, since TABLETOUT sends an appropriate value here automatically (see TABLETOUT).
4. The Boolean value on output <2> indicates the condition of the stylus tipswitch (or cursor button) as follows:

   TRUE = Stylus tipswitch closed or cursor button pressed.
   FALSE = Stylus tipswitch open or cursor button not pressed.

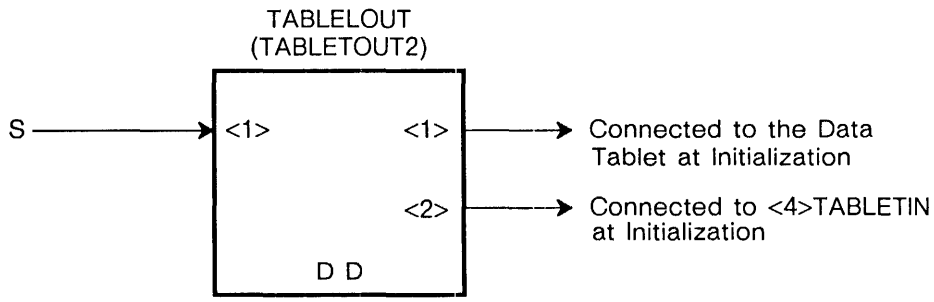
5. The integer on output <3> is the sum of the numbers of the pressed buttons on the 4-button cursor. The buttons are numbered 1, 2, 4, and 8. If button 1 and button 4 are pressed simultaneously, 5 is output.

6. A TRUE appears at output <4> whenever the tipswitch goes from open to closed, and a FALSE whenever the tipswitch goes from closed to open. For button-type cursors, output <4> is TRUE when a button is pushed and FALSE when the button is released.
7. Output <5> indicates transitions in stylus proximity (i.e., from "receiving data" to "not receiving data" and vice versa). A TRUE appears here when data is received from the tablet after a period of no data. A FALSE is sent when data does not arrive from the tablet in time. The time is the number of hundredths of a second specified at input <4>.
8. Output <6> is the (X,Y)-position of the stylus when the tipswitch goes from open to closed.
9. Inputs <5> and <6> allow the user to redefine the origin of the tablet.

TABLETOUT

TYPE

Initial Function Instance — Data Input



PURPOSE

Provides the means to set operating parameters in the data tablet by sending a character to input <1>. The character also determines a value to be sent to <4>TABLETIN, setting the timeout interval of the tablet. If a multicharacter string is sent to input <1>, only the final character of the string is used.

DESCRIPTION

INPUT

<1> — character or string

OUTPUTS

<1> — connected to data tablet

<2> — connected to <4>TABLETIN

NOTES

1. Characters for mode settings are shown in the following table.

CHARACTER	MODE	SAMPLING RATE	TIMEOUT INTERVAL (IN CENTISECONDS)	
S	Stop	Idle		
P	Point*	Manual control		
@	Switched Stream**	2	52	
A	.	4	27	
B	.	10	12	
C	.	20	8	
D	.	35	5	
E	Switched Stream	70	3	
H	Stream***	2	52	
I	.	4	27	
J	.	10	12	
K	.	20	8	(default)
L	.	35	5	
M	Stream	70	3	

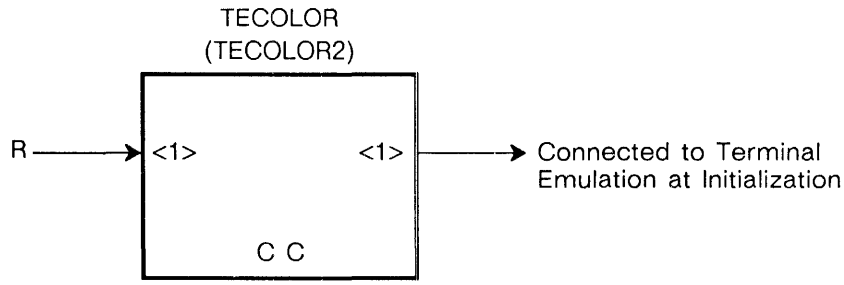
* Pressing the stylus on the tablet or the button on the cursor sends out the single X,Y coordinate pair.

** Pressing the stylus on the tablet surface or the button on the cursor causes X,Y coordinate pairs to be output continuously at the selected sampling rate until the stylus is lifted or the cursor button is released.

*** X,Y coordinate pairs are generated continuously at the selected sampling rate when the stylus or cursor is in the proximity of the tablet surface. Pressing the stylus on the tablet surface or pressing the cursor button sets the flag character (F) in the output stream.

TYPE

Initial Function Instance — Miscellaneous



PURPOSE

Specifies the hue of terminal emulator and setup output.

DESCRIPTION

INPUT

<1> — hue

OUTPUT

<1> — connected to terminal emulator

DEFAULT

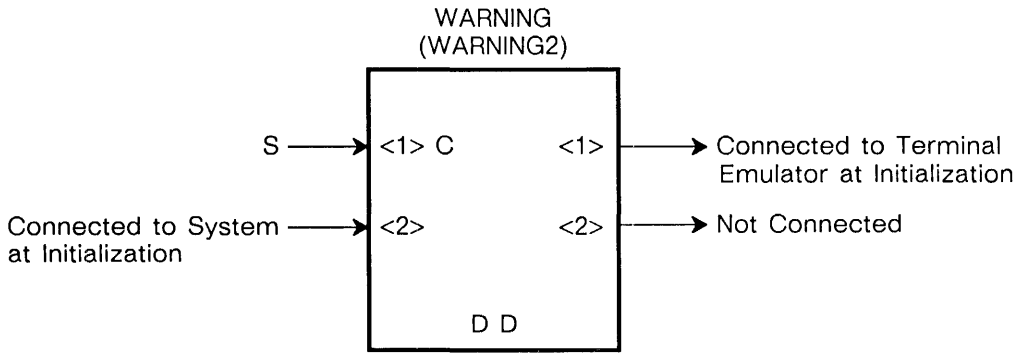
The default hue is 240, pure green.

NOTES

The range of acceptable values is the 0–360 "color wheel" used by the SET COLOR command, in which 0 represents pure blue, 120 pure red, and 240 pure green. The default is 240. Out-of-range values are clamped to the nearest in-range value (0 or 360 - hence always blue).

TYPE

Initial Function Instance — Miscellaneous



PURPOSE

Enables and disables the display of warning messages.

DESCRIPTION

INPUTS

<1> — enable/disable warning messages (constant)

<2> — connected to system

OUTPUTS

<1> — connected to terminal emulator

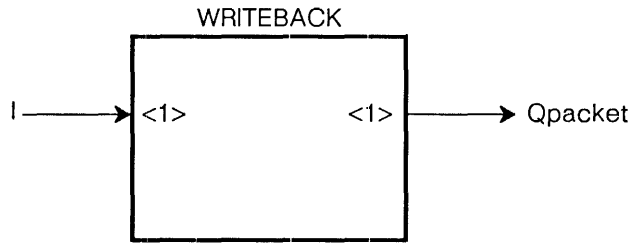
<2> — not used

NOTE

A TRUE at input <1> enables warning messages. A FALSE at input <1> disables them. The INITIALIZE command automatically sends a TRUE to input <1> enabling the display of warning messages.

TYPE

Initial Function Instance — Miscellaneous



PURPOSE

WRITEBACK is initialized by the system and is used to send encoded writeback data to user function networks.

This function is not activated by the normal input queue triggering mechanism. It is activated by sending a TRUE to any WRITEBACK operation node.

DESCRIPTION

INPUT

<1> — integer specifying the size of Qpackets to be output by the function

OUTPUT

<1> — passes the encoded writeback data as Qpackets

DEFAULT

The default size on input <1> is 512. Minimum and maximum sizes are 16 and 1024. If the size specified on the input is not within this range, the default size will be used.

NOTES

1. WRITEBACK will return all data that is under the WRITEBACK operation node. Host-resident code will be responsible for recognizing the start-of-writeback and end-of-writeback commands. Attribute information, such as color, must be interpreted by host code to ensure that the hardcopy plots are correct.

2. On the PS 390, viewport translations have not been applied to the data. To correctly compute the position of endpoints, the host program interpreting the writeback code must add a viewport center to each endpoint. The initial viewport center is established with a VIEWPORT CENTER command. The VIEWPORT CENTER command is sent following the start-of-writeback command. Any changes to the viewport center will be indicated through this sequence of commands: CLEAR DDA, CLEAR SAVE POINT, position endpoint, CLEAR SAVE POINT. The position endpoint becomes the new viewport center.

CURSOR

TYPE

Initial Structure

CURSOR
(CURSOR2)

```
Cursor := VECTOR_LIST ITEMIZED N = 4
          P .035, .035 L - .035, - .035
          P -.035, .035 L .035, -.035;
```

PURPOSE

This initial structure is a vector list as shown above, which creates a displayable cursor in the form of a cross when the system is initialized.

DESCRIPTION

INPUT

Vector List

OUTPUT

Displayable "X"-shaped cursor

NOTES

1. The cursor is controlled by a function network which positions it on the PS 390 screen in response to stylus movement over the data tablet surface. The intensity of the cursor increases when the stylus tip switch is pressed down.
2. The user is free to redefine CURSOR using any other vector list.

PICK_LOCATION

TYPE

Initial Structure

PICK_LOCATION
(PICK_LOCATION2)

PURPOSE

PICK_LOCATION is the name assigned at initialization to the system-created picking location.

DEFAULT

At system initialization, the pick location is defined as the center of the cursor.

NOTE

The initial TABLETIN function instance is connected to PICK_LOCATION and the system-initialized CURSOR points to its center.

Appendix A
Initial Function Instances by Category

- **Data Input**

  BUTTONSIN
  DIALS
  FKEYS
  KEYBOARD
  MOUSEIN
  PICK
  SPECKEYS
  TABLETIN
  TABLETOUT

- **Data Output**

  CLEAR_LABELS
  DLABEL1...DLABEL8
  DSET1...DSET8
  FLABEL0
  FLABEL1...FLABEL12
  HOSTOUT
  OFFBUTTONLIGHTS
  ONBUTTONLIGHTS

- **Miscellaneous**

  ERROR
  HOST_MESSAGE
  HOST_MESSAGEB
  INFORMATION
  MEMORY_ALERT

- **Miscellaneous** *(continued)*

  MEMORY_MONITOR
  MESSAGE_DISPLAY
  PS390ENV
  SHADINGENVIRONMENT
  TECOLOR
  WARNING
  WRITEBACK

RM4. GRAPHICS SUPPORT ROUTINES

RM4. GRAPHICS SUPPORT ROUTINES

CONTENTS

1. PS 390 GSR Error Code Definitions 2

PAttch
PAttach 8
PAttr
PAttrib 11
PAttr2
PAttrib2 13
PBeg
PBegin 16
PBegS
PBeginS 17
PBspl 18
PChRot
PCharRot 21
PChs
PChars 22
PChSca
PCharSca 24
PConn
PConnect 26
PCopyV
PCopyVec 28
PDefPa
PDefPatt 30
PDelet
PDelete 32
PDelim 33
PDeLOD
PDecLOD 34
PDelW
PDelWild 35

PDi
PDisc 36
PDiAll
PDiscAll 38
PDInfo
PDevInfo 39
PDiOut
PDiscOut 40
PDisp
PDisplay 41
PDtach
PDetach 42
PEnd 43
PEndOp
PEndOpt 44
PEndS 45
PEraPa
PEraPatt 46
PEyeBk
PEyeBack 47
PFn
PFnInst 49
PFnN
PFnInstN 50
PFoll 52
PFont 53
PForg
PForget 54
PFov 55
PGet 57
PGetW
PGetWait 59
PGUCPU
PGiveUpCPU 61
PIfBit 62
PIfLev
PIfLevel 64
PIfPha
PIfPhase 66
PIllum
PIllumin 68
PIncl 70

PInit 71
PInitC 72
PInitD 73
PInitN 74
PInLOD
PIncLOD 75
PInst 76
PLaAdd
PLabAdd 77
PLaBeg
PLabBegn 79
PLaEnd
PLabEnd 81
PLoad 82
PLookA
PLookAt 83
PMat22
PMat2x2 85
PMat33
PMat3x3 86
PMat43
PMat4x3 87
PMat44
PMat4x4 89
PMuxCI 90
PMuxG 91
PMuxP
PMuxPars 92
PNil
PNameNil 93
POpt
POptStru 94
PPatWi
PPatWith 95
PPlygA
PPlygAtr 96
PPlygB
PPlygBeg 97
PPlygE
PPlygEnd 99
PPlygH
PPlygHSI
PPlygLisHSI 100

PPlygL
PPlygLis 104
PPlygO
PPlygOtl 107
PPlygR
PPlygRGB
PPlygLisRGB 109
PPoly 113
PPref 115
PPurge 116
PPutG 117
PPutGX 118
PPutP
PPutPars 119
PRasCp 120
PRasEr 122
PRasLd 124
PRasWP 126
PRawBl
PRawBlo 128
PRaWRP 129
PRBspl 130
PRem 133
PRemFo
PRemFoll 134
PRemFr
PRemFrom 135
PRemPr
PRemPref 136
PRotX 137
PRotY 138
PRotZ 139
PRPoly 140
PRsvSt
PRsvStor 143
PSavBeg 144
PSavEnd 145
PScale
PScaleBy 146
PSeBit
PSetBit 147
PSeChF
PSetChrF 149

PSeChS
PSetChrS 151
PSeChW
PSetChrW 153
PSeCns
PSetCnes 155
PSeCol
PSetColr 156
PSeCon
PSetCont 158
PSecPl
PSecPlan 159
PSeDAl
PSetDAll 160
PSeDCL
PSetDCL 161
PSeDOF
PSetDOnF 163
PSeInt
PSetInt 164
PSeLnt
PSetLinT 166
PSeLOD
PSetLOD 169
PSePID
PSetPID 171
PSePLo
PSetPLoc 172
PSePOf
PSetPOnf 174
PSeR
PSetR 175
PSeREx
PSetRExt 177
PSnBoo
PSndBool 178
PSnFix
PSndFix 180
PSnM2d
PSndM2d 182
PSnM3d
PSndM3d 184

PSnM4d
PSndM4d 186
PSnPL
PSndPL 188
PSnRea
PSndReal 190
PSnRSt
PSndRStr 191
PSnSt
PSndStr 193
PSnV2d
PSndV2d 195
PSnV3d
PSndV3d 197
PSnV4d
PSndV4d 199
PSnVal
PSndVal 201
PSnVL
PSndVL 203
PSolRe
PSolRend 205
PStdFo
PStdFont 206
PSURGB
PSUTIL_HSIRGB 207
PSurRe
PSurRend 208
PTrans
PTransBy 209
PVar 211
PVcBeg
PVecBegn 212
PVcEnd
PVecEnd 215
PVcLis
PVecList 216
PVcMax
PVecMax 219
PViewP 220
PWindo
PWindow 222

PWrtBk
PWrtBack 224
PXfCan
PXfCancl 225
PXfMat
PXfMatrx 226
PXfVec
PXfVectr 227

Appendix A
GSRs and Corresponding ASCII Commands 228

TABLES

Table 4-1. Warning Codes 2
Table 4-2. Command Error Codes 3
Table 4-3. User Error Codes 4
Table 4-4. Parsing Error Codes 6
Table 4-5. Fatal Error Codes 7
Table 4-6. Internal GSR Validity Fatal Error Codes 7

Section RM4
Graphics Support Routines

This section contains a listing of the VAX FORTRAN and Pascal, IBM FORTRAN and Pascal, and UNIX/C Graphics Support Routines (GSRs), and error tables which define the possible error codes used to identify warning, error or fatal error conditions that may arise while using the GSRs. Section *TT3 Using the GSRs* discusses GSR conventions and definitions, error handling, and programming suggestions.

Utility Routines and Application Routines are the two types of GSRs. Utility Routines are specific to the operation of the GSRs, and are used to attach the PS 390, set the string delimiting character, select multiplexing channels, send and receive messages, and detach. Application Routines correspond almost one for one with the standard PS 390 commands. The raster routines do not have a corresponding ASCII command. In most cases, the names for the Application Routines were derived by choosing an abbreviation of the PS 390 commands and prefixing it with a P. Parameter ordering generally coincides with the PS 390 command as well. In this section, the Utility and Application Routines for each language and operating system are grouped together. They are listed in alphabetical order according to the FORTRAN GSR. Appendix A lists the GSRs and the corresponding ASCII command or utility or raster routine name.

The upper left corner lists the PS 390 command if it is an Application Routine or notes that it is a Utility Routine. The upper right corner lists the name of the GSR for the FORTRAN, Pascal and UNIX/C languages in that order. The name of the GSR in a particular language is listed only once if it is identical to another language.

The GSRs are listed in the following order: VAX and IBM FORTRAN, VAX Pascal, IBM Pascal, and UNIX/C. The names of the FORTRAN GSRs are limited to six characters, and there is no character limit for the names of the Pascal and UNIX/C GSRs. The UNIX/C GSRs are case-sensitive, and the FORTRAN and Pascal GSRs are not case-sensitive. Note that parameter names, not variable names, are listed in the GSRs.

Following the listing of the GSRs, a description summarizes the purpose and operation of the GSR, and contains comments and notes. The PS 390 Command

and Syntax is listed for the Application Routines. Cross references to related GSRs are listed under SEE ALSO. Cross reference entries are almost exclusively to PS 390 commands and not to the name of the GSR.

1. PS 390 GSR Error Code Definitions

The following tables define the possible error codes used to identify warning or error conditions that may arise while using the Graphics Support Routines. The set of possible error codes is divided into several regions reserved for specific severity and machine dependency levels. When there is a difference in routine names, the Pascal name is given in parentheses. These error codes apply to DEC VAX/VMS FORTRAN, DEC VAX/VMS Pascal, IBM FORTRAN and IBM Pascal GSRs.

```
   1...255 = Machine INDEPENDENT warning conditions.
 256...511 = Machine   DEPENDENT warning conditions.
 512...767 = Machine INDEPENDENT error conditions.
 768...1023 = Machine   DEPENDENT error conditions.
1024...1279 = Machine INDEPENDENT fatal error conditions.
1280...1535 = Machine   DEPENDENT fatal error conditions.
```

The following warning codes allow successful completion of the GSR, but indicate a probable user error.

Table 4-1. Warning Codes

Error Code	Mnemonic	Meaning
1	PSWBNC PSW_BadNamChr	Bad name character. Any invalid PS 390 name character is translated to the underscore character.
2	PSWNTL PSW_NamTooLon	Name too long. Name truncated to 256 characters.
3	PSWSTL PSW_StrTooLon	String too long. String truncated to 240 characters.
30	PSWPCG PSW_PixCouGre	The Pixel Count is greater than the screen size in call to PRASWP. (Reserved for P6.V01 Raster routines.)
31	PSWPCL PSW_PixCouLes	The Pixel Count is less than 1 in call to PRASWP. (Reserved for P6.V01 Raster routines.)
32	PSWRCG PSW_RepCouGre	Repetition count greater than 255 in call to PRASLU. (Reserved for P6.V01 Raster routines.)

Table 4-1. (continued)

Error Code	Mnemonic	Meaning
33	PSWRCL PSW_RepCouLes	Repetition count less than 1 in call to PRASLU. (Reserved for P6.V01 Raster routines.)
256	PSWAAD PSW_AttAlrDon	Attach already done.
257	PSWAKS PSW_AtnKeySee	Attention key seen. This tells the error-handling routine that the user hit the Attention key (IBM version only).
258	PSWBGC PSW_BadGenChr	The string specified to be sent to the "generic" output channel of CIROUTE via the PPutGX routine contained an invalid character that has been translated to a blank space character. This error code cannot be caused by invoking the routine PPutG which does not perform any translation on the specified string (IBM version only).
259	PSWBSC PSW_BadStrChr	Bad string character. Any invalid string character is converted to a blank space character.
260	PSWBPC PSW_BadParChr	The string specified to be sent to the PS 390 Parser via the PPutP (PPutPars) routine contained an invalid character that has been translated to a blank space character.

For the following errors, the GSRs abort the current command sequence (if there is one) and ignore the out-of-sequence command that (probably) caused this error.

Table 4-2. Command Error Codes

Error Code	Mnemonic	Meaning
515	PSEPOE PSE_PreOpeExp	Prefix operate node call expected.
516	PSEFOE PSE_FolOpeExp	Follow operate node call expected.
517	PSELBE PSE_LabBlkExp	Label block call expected.

Table 4-2. Command Error Codes (continued)

Error Code	Mnemonic	Meaning
518	PSEVLE PSE_VecLisExp	Vector List call expected.
519	PSEAMV PSE_AttMulVec	Attempted multiple PVcLis (PVecList) call sequence for block normalized vectors prohibited.
520	PSEMLB PSE_MisLabBeg	Missing label block begin call.
521	PSEMVB PSE_MisVecBeg	Missing vector list begin call.
529	PSEMPB PSE_MisPolBeg	The Begin polygon call is missing. PPLYGA (PPlygAtr), PPLYGL (PPlygLis), or PPLYGE (PPlygEnd) was called without the prerequisite call to PPlygB (PPlygBeg).
530	PSEALE PSE_PAtPliPen	A call to PPLYGA (PPlygAtr), PPLYGL (PPlygLis), or PPLYGE (PPlygEnd) was expected.
531	PSELEX PSE_PLiPEnExp	A call to PPLYGL (PPlygLis) or PPLYGE (PPlygEnd) was expected.
532	PSEALX PSE_PAtPLiExp	A call to PPLYGA (PPlygAtr) or PPLYGL (PPlygLis)was expected.
533	PSELX PSE_PLiExp	A call to PPLYGL (PPlygLis) was expected.

The following errors are user errors and are generated by invalid parameters or by an unsuccessful attempt to attach.

Table 4-3. User Error Codes

Error Code	Mnemonic	Meaning
512	PSEIMC PSE_InvMuxCha	Invalid multiplexing channel argument specified a call to PMuxP (PMuxPars), PmuxCI (PMuxCI), or PMuxG. The multiplexing channel assigned to the Parser, CI, or Generic channel is not changed.
513	PSEIVC PSE_InvVecCla	Invalid vector list class specified in call to PVcBeg (PVecBegn). Command is ignored.

Table 4-3. User Error Codes (continued)

Error Code	Mnemonic	Meaning
514	PSEIVD PSE_InvVecDim	Invalid vector list dimension specified in call to PVcBeg (PVecBegn). Command is ignored.
522	PSENUN PSE_NulNam	A null name is not permitted in this call context. The command is ignored.
523	PSEBCT: PSE_BadComTyp	Bad Comparison type operator specified. If Level = command ignored.
524	PSEIFN PSE_InvFunNam	Attempted PS 390 function instance call failed because the named function cannot possibly exist. The function name identifying the function type to instance was longer than 256 characters.
525	PSENNR PSE_NulNamReq	Null name was required for parameter in operate node call following a PPref or PFoll routine.
526	PSETME PSE_TooManEnd	Too many PEndS calls for the number of preceding PBEGS (PBeginS) calls. Command ignored.
527	PSENOA PSE_NotAtt	The PS 390 communications link has not been established. The user failed to call PAttch (PAttach) or an error occurred in the attach routine preventing the communications link from being created.
528	PSEODR PSE_OveDurRea	An overrun occurred during a read operation. The user-supplied input buffer was too small and truncation has occurred.
534	PSEMPX PSE_MaxPolExc	The polygon specified by the call to PPLYGL (PPlygLis) contains more than 250 vertices. The polygon is ignored.
535	PSELMP PSE_LesMinPol	The polygon specified by the call to PPLYGL (PPlygLis) contains fewer than 3 vertices. It is therefore a degenerate polygon and is ignored.
536	PSEIPA PSE_IllPolAtr	Illegal polygon attribute(s) specified in the call to PPLYGA (PPlygAtr). The attribute(s) are ignored.
550	PSEICP PSE_IllCurPix	Illegal Current Pixel specification in call to PRASCP. (Reserved for P6.V01 Raster routines.)

Table 4-3. User Error Codes (continued)

Error Code	Mnemonic	Meaning
552	PSEIOR PSE_IndOutRan	Index out of range: 0...255 in call to PRASLU. (Reserved for P6.V01 Raster routines.)
553	PSELDC PSE_IllDCpe	Illegal LDC specification in call to PRASLD. (Reserved for P6.V01 Raster routines.)
554	PSELNL PSE_SLUNumLes	NUM parameter less than 1 in call to PRASLD. (Reserved for P6.V01 Raster routines.)
555	PSEMGM PSE_MinGreMax	Minimum > Maximum in call to PRASLR. (Reserved for P6.V01 Raster routines.)
556	PSEMNO PSE_MinOutRan	Minimum out of range 0...255 in call to PRASLR. (Reserved for P6.V01 Raster routines.)
557	PSEMXO PSE_MaxOutRan	Maximum out of range 0...255 in call to PRASLR. (Reserved for P6.V01 Raster routines.)
558	PSEPNL PSE_SWPNumLes	NUM parameter less than 1 in call to PRASWP. (Reserved for P6.V01 Raster routines.)

At the present time, the following three error messages (780, 781, 782) are only meaningful for Digital Equipment Corporation (DEC) VAX/VMS* (*Trademark of the Digital Equipment Corporation, Maynard, Massachusetts). All three errors indicate that the parameter passed as a string in PAttch (PAttach) was not successfully parsed and that the Attach call failed.

Table 4-4. Parsing Error Codes

Error Code	Mnemonic	Meaning
780	PSEPDT PSE_PhyDevTyp	This error indicates that a missing or invalid Physical Device Type was specified in a call to PAttch (PAttach).
781	PSELDN PSE_LogDevNam	This error indicates that a missing or invalid Logical Device Name was specified in a call to PAttch (PAttach).
782	PSEADE PSE_AttDelExp	This error indicates that an Attach delimiter was expected in a call to PAttch (PAttach).

The errors listed below indicate a very serious error condition. If the user's error handler is invoked with any of the error codes listed below, the program execution should be aborted.

Table 4-5. Fatal Error Codes

Error Code	Mnemonic	Meaning
1024	PSFIFC PSF_IllFraCom	Illegal frame command specified in call to PSUTIL_RasMode. This error code indicates an internal validity check error. E&S Software Support should be contacted.
1280	PSFPAF PSF_PhyAttFai	Physical Attach operation failed.
1281	PSFPDF PSE_PhyDetFai	Physical Detach operation failed.
1282	PSFPGF PSF_PhyGetFai	Physical Get operation failed.
1283	PSFPPF PSE_PhyPutFai	Physical Put operation failed.

The following three errors are only applicable to the DEC VAX/VMS version of the Graphics Support Routines. All three error codes indicate an internal Graphics Support Routines validity error. E&S Software Support should be contacted if these errors are detected.

Table 4-6. Internal GSR Validity Fatal Error Codes

Error Code	Mnemonic	Meaning
1290	PSFBTL PSF_BufTooLar	Buffer too large in a call to PSPUT. Internal validity check error.
1291	PSFWNA PSF_WroNumArg	Wrong number of arguments to low-level I/O routine in PROIOLIB.MAR. Internal validity check error.
1292	PSFPTL PSF_ProTooLar	Prompt too large in call to PSFRCV. Internal validity check error.

IBM FORTRAN UTILITY ROUTINE

```
CALL PAttch (Modifiers, ErrHnd)
```

where:

Modifiers is a CHARACTER STRING reserved for future use by the Evans & Sutherland Computer Corporation. It is currently ignored by the IBM GSRs.
ErrHnd is the user-defined error-handler subroutine.

IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PAttach (  CONST Modifiers : P_VaryingType;
                     PROCEDURE Error_Handler (Error : INTEGER));
```

where:

Modifiers is a CHARACTER STRING reserved for future use by the Evans & Sutherland Computer Corporation. It is currently ignored by the IBM GRSs.
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This routine attaches the PS 390 to the communications channel. If it is not called prior to use of the application routines, the error code value corresponding to PSENOA (PSE_NotAtt — Pascal) is generated, indicating that the PS 390 communications link has not been established.

VAX FORTRAN UTILITY ROUTINE

```
CALL PAttch (Modifiers, ErrHnd)
```

VAX PASCAL UTILITY ROUTINE

```
PROCEDURE PAttach (   %DESCR Modifiers : P_VaryingType;
                    PROCEDURE Error_Handler (Error : INTEGER));
```

DESCRIPTION

This procedure attaches the PS 390 to the communications channel. If this procedure is not called prior to use of the Application Procedures, the error

code value corresponding to the name, PSENOA (PSE_NotAtt — Pascal), is generated, indicating that the PS 390 communications link has not been established. The parameter, **Modifiers**, must contain the phrases:

LOGDEVNAM=name/PHYDEVTYP=type

where 'name' refers to the logical name of the device that the GSRs will communicate with, i.e. TTA6:, TTB2: XME0:, PS:, etc. and 'type' refers to the physical device type of the hardware interface that the GSRs will communicate through. This last argument can only be one of the following three interfaces:

ASYNC (standard RS-232 asynchronous communication interface)
PARALLEL (Parallel interface option)
ETHERNET (DECnet Ethernet option)

The parameter string must contain EXACTLY one "/" and blanks are NOT allowed to surround the "=" in the phrases. The Pattach parameter string is not sensitive to upper or lower case.

Example: PAttach (´logdevnam=tta2:/phydevtyp=async´, Error_Handler);

where 'tta2:' is the logical device name of the PS 390, and the hardware interface is standard asynchronous RS-232.

Example: PAttach (´logdevnam=ps:/phydevtyp=ethernet´, Error_Handler);

where the physical device type is an Ethernet interface, and where the user has informed the VAX that the logical symbol 'ps:' refers to the name of the logical device that the GSRs will communicate with using the following ASSIGN command:

$ ASSIGN XMD0: PS

$ RUN <application-pgm>

UNIX/C UTILITY ROUTINE

```
#include <ps300/gsrext.h>

PAttach( devname )

where:

    string devname;
```

DESCRIPTION

PAttach establishes a communication channel between a UNIX process and the PS 390. The **devname** parameter is a character string specifying the communication channel. The following channels are supported:

ASYNC (RS-232 asynchronous interface)
To use the asynchronous serial communications channel to the PS 390, **devname** should be "-" in the case of **stdout** and "/dev/ttyxx" otherwise.

PARALLEL (parallel interface option)
To use the Unibus Parallel Interface device, **devname** should be a string of the form "/dev/pixy", where **pixy** is a special file referring to a parallel interface device unit. Check with your system manager to get the name(s) of the Parallel Interface special file(s) for your system.

ETHERNET (Ethernet interface option)
To use the Ethernet interface, **devname** should be a string representing the network node as listed in the /etc/hosts file.

PAttach returns 0 (Zero) on successful attach and 1280 (PSF_PhyattFai) on unsuccessful attach. The PAttach function must be called prior to any other routine in the GSR library.

SEE ALSO

PDtach, PDetach

VAX and IBM FORTRAN GSR

```
CALL PAttr (Name, Hue, Saturation, Intensity, Opaque, Diffused,
            Specular, ErrHnd)
```

where:

Name is a CHARACTER STRING
Hue is a REAL
Saturation is a REAL
Intensity is a REAL
Opaque is a REAL
Diffused is a REAL
Specular is an INTEGER*4
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PAttrib ( %DESCR Name         : P_Varying_Type;
                           Hue          : REAL;
                           Saturation   : REAL;
                           Intensity    : REAL;
                           Opaque       : REAL;
                           Diffused     : REAL;          {default .75}
                           Specular     : REAL;          {default 4}
                    Procedure Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PAttrib ( CONST  Name         : STRING;
                           Hue          : REAL;
                           Saturation   : REAL;
                           Intensity    : REAL;
                           Opaque       : REAL;
                           Diffused     : REAL;          {default .75}
                           Specular     : REAL;          {default 4}
                    Procedure Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include<ps300/gsrext.h>
PAttrib (name,hue,saturation,intensity,opaque,diffused,specular)
```

where:

```
string name;
double hue,saturation,intensity,opaque,diffused;
integer specular;
```

DESCRIPTION

This routine defines polygon characteristics used by the rendering firmware in the PS 390 to produce shaded renderings. **Hue, saturation** and **intensity** define the color of the polygon. **Hue** specifies an angle between 0 and 360 indicating the color on a color wheel with pure blue being 0, red being 120 and green being 240. **Saturation** specifies the saturation of the color with 0 being no color and 1 being full saturation. **Intensity** specifies the intensity of the color with 0 being no color (black) and 1 being full intensity. **Opaque** specifies how transparent the polygon is with 1 being fully opaque and 0 being fully transparent. **Diffused** is the proportion of color contributed by diffuse reflection versus that contributed by specular reflection with a value of 1 eliminating all specular highlighting and a value of 0 eliminating all diffuse reflectivity. **Specular** adjusts the concentration of specular highlights in the range of 0 to 10.

PS 390 Command and Syntax

```
Name  := ATTRIBUTES [COLOR h[,s[i]]] [DIFFUSE d] [SPECULAR s] [OPAQUE t];
```

SEE ALSO

POLYGON

VAX and IBM FORTRAN GSR

```
CALL PAttr2 (Name, Hue, Saturation, Intensity, Opaque, Diffused,
             Specular, Hue2, Saturation2, Intensity2, Opaque2,
             Diffused2, Specular2, ErrHnd)
```

where:

Name is a CHARACTER STRING
Hue is a REAL
Saturation is a REAL
Intensity is a REAL
Opaque is a REAL
Diffused is a REAL
Specular is an INTEGER*4
Hue2 is a REAL
Saturation2 is a REAL
Intensity2 is a REAL
Opaque2 is a REAL
Diffused2 is a REAL
Specular2 is an INTEGER*4
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PAttrib2 ( %DESCR Name        : P_Varying_Type;
                            Hue         : REAL;
                            Saturation  : REAL;
                            Intensity   : REAL;
                            Opaque      : REAL;
                            Diffused    : REAL;          {default .75}
                            Specular    : REAL;          {default 4}
                            Hue2        : REAL;
                            Saturation2 : REAL;
                            Intensity2  : REAL;
                            Opaque2     : REAL;
                            Diffused2   : REAL;          {default .75}
                            Specular2   : REAL;          {default 4}
                     Procedure Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PAttrib2 ( CONST  Name         : STRING;
                            Hue          : REAL;
                            Saturation   : REAL;
                            Intensity    : REAL;
                            Opaque       : REAL;
                            Diffused     : REAL;          {default .75}
                            Specular     : INTEGER;       {default 4}
                            Hue2         : REAL;
                            Saturation2  : REAL;
                            Intensity2   : REAL;
                            Opaque2      : REAL;
                            Diffused2    : REAL;          {default .75}
                            Specular2    : INTEGER;       {default 4}
                     Procedure Error_Handler (Err : INTEGER));
```

C UNIX/C GSR

```
#include <ps300/gsrext.h>
PAttrib2 (name,hue1,saturation1,intensity1,opaque1,diffused1,specular1,
          hue2,saturation2,intensity2,opaque2,diffused2,specular2)
```

where:

```
string name;
double hue1,saturation1,intensity1,opaque1,diffused1;
integer specular1;

string name;
double hue2,saturation2,intensity2,opaque2,diffused2;
integer specular2;
```

DESCRIPTION

This routine defines polygon characteristics used by the rendering firmware in the PS 390 to produce shaded renderings. This routine allows for a second set of attributes to be defined for the backside of polygons.

PS 390 Command and Syntax

```
Name  := ATTRIBUTES [COLOR h[,s[i]]] [DIFFUSE d] [SPECULAR s] [OPAQUE t];

          AND       [COLOR h2[,s2[i2]]] [DIFFUSE d2] [SPECULAR s2]
                    [OPAQUE t2];
```

VAX and IBM FORTRAN GSR

```
CALL PBeg (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PBegin (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PBegin (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PBegin();
```

DESCRIPTION

This routine is used with the END routine to group a set of viewing and/or modeling commands so that they appear to be executed simultaneously.

PS 390 Command and Syntax

```
Name := BEGIN...END;
```

SEE ALSO

END

VAX and IBM FORTRAN GSR

```
CALL PBegS (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PBeginS (  %DESCR Name : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PBeginS (  CONST Name : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PBeginS( name );
```

where:

string name;

DESCRIPTION

This routine is used with the END STRUCTURE routine to group a set of viewing and/or modeling commands so that each element does not need to be explicitly named.

PS 390 Command and Syntax

```
Name := BEGIN_Structure...END_Structure;
```

SEE ALSO

END_STRUCTURE

VAX and IBM FORTRAN GSR

```
CALL PBspl (Name, Order, OpenClosed, NonPeriodic_Periodic, Dimension,
            Nvertices, Vertices, KnotCount, Knots, Chords, ErrHnd)
```

where:

Name is a CHARACTER STRING
Order is an INTEGER*4 specifying the order of the B-spline
OpenClosed is a LOGICAL*1
NonPeriodic_Periodic is a LOGICAL*1
Dimension is an INTEGER*4
Nvertices is an INTEGER*4 specifying the number of vertices
Vertices is defined: REAL*4 Vertices (4, NVertices) specifying the vertices

where:
```
Vertex (1,n) = x (n)
Vertex (2,n) = y (n)
Vertex (3,n) = z (n)
Vertex (4,n) is not used.
```

KnotCount is an INTEGER*4 specifying the number of knots
Knots is an array (KnotCount+1) of REAL*4 specifying the knot sequence
Chords is an INTEGER*4 specifying the number of vectors to be created
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PBspl (   %DESCR Name                 : P_VaryingType;
                           Order                : INTEGER;
                           OpenClosed           : BOOLEAN;
                           NonPeriodic_Periodic: BOOLEAN;
                           Dimension            : INTEGER;
                           N_Vertices           : INTEGER;
                       VAR Vertices             : P_VectorListType;
                           KnotCount            : INTEGER;
                       VAR Knots                : P_KnotArrayType;
                           Chords               : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PBspl (   CONST Name                  : STRING;
                          Order                 : INTEGER;
                          OpenClosed            : BOOLEAN;
                          NonPeriodic_Periodic: BOOLEAN;
                          Dimension             : INTEGER;
                          N_Vertices            : INTEGER;
                    CONST Vertices              : P_VectorListType;
                          KnotCount             : INTEGER;
                    CONST Knots                 : P_KnotArrayType;
                          Chords                : INTEGER;
                PROCEDURE Error_Handler  (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PBspl(name,order,openclosed,nonperiodic_periodic,dimension,n_vertices,
      vertices, knotcount, knots,chords )
```

where:

```
string name;
integer order,dimension,n_vertices,knotcount,chords;
boolean openclosed,nonperiodic_periodic;
P_VectorListType vertices;
P_KnotArrayType knots;
```

DESCRIPTION

This routine evaluates a B-spline curve, allowing the parametric description of the curve form without having to specify the coordinates of each vector.

In the parametric definitions:

- **Name** specifies the name to be assigned to the computed B-spline.
- **Order** is the order of the curve.
- **Openclosed** is TRUE for open and FALSE for closed.
- **NonPeriodic_Periodic** is TRUE for nonperiodic and FALSE for periodic.
- **Dimension** is 2 or 3 (2 or 3 dimensions respectively).

- **N_Vertices** specifies the number of vertices.
- **Vertices** specifies the vertices of the B-spline.
- **Knotcount** specifies the number of knots.
- **Knots** specifies the knot sequence to be used in computing the B-spline.
- **Chords** is the number of vectors to be created.

NOTE

None of the parameters in the routine are optional. If Knotcount = 0, then the default knot sequence is generated and the knot array is ignored. In the PS 390 command, dimension is implied by the syntax.

PS 390 Command and Syntax

```
Name  := BSPLINE ORDER = k [OPEN/CLOSED] [NONPERIODIC/PERIODIC] [N = n]
         [VERTICES =] X1,Y1,Z1
                      X2,Y2,Z2
                       :  :  :
                      Xn,Yn,Zn
         [KNOTS] = t1,t2,...,tj
         CHORDS = q;
```

SEE ALSO

RATIONAL BSPLINE, POLYNOMIAL, RATIONAL POLYNOMIAL

VAX and IBM FORTRAN GSR

```
CALL PChRot (Name, Angle, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PCharRot (   %DESCR Name       : P_VaryingType;
                              Angle      : REAL;
                       %DESCR AppliedTo  : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PCharRot (   CONST  Name       : STRING;
                              Angle      : SHORTREAL;
                       CONST  AppliedTo  : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PCharRot( name,angle,appliedto )
```

where:

string name,appliedto;
double angle;

DESCRIPTION

This routine rotates the specified characters (Apply/AppliedTo), where **Angle** is the Z-rotation angle in degrees.

PS 390 Command and Syntax

```
Name := CHARacter ROTate Angle [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PChs (Name, TranX, TranY, TranZ, StepX, StepY, Chars, ErrHnd)
```

where:

Name is a CHARACTER STRING
TranX, TranY, TranZ are REAL*4
StepX, StepY are REAL*4
Chars is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PChars (   %DESCR Name       : P_VaryingType;
                            TranX      : REAL;
                            TranY      : REAL;
                            TranZ      : REAL;
                            StepX      : REAL;
                            StepY      : REAL;
                     %DESCR Chars      : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PChars (   CONST  Name       : STRING;
                            TranX      : SHORTREAL;
                            TranY      : SHORTREAL;
                            TranZ      : SHORTREAL;
                            StepX      : SHORTREAL;
                            StepY      : SHORTREAL;
                     CONST  Chars      : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PChars( name,tranx,trany,tranz,stepx,stepy,chars )
```

where:

string name,chars;
double tranx,trany,tranz,stepx,stepy;

DESCRIPTION

This routine defines a character string **Chars** and specifies its location and placement. It has the following parametric definitions:

TranX, **TranY**, and **TranZ** give the x,y,z coordinates of the location of the beginning of the character string.

StepX, **StepY** give the spacing between characters in character unit size.

PS 390 Command and Syntax

```
Name := CHARacters [x,y[,z][STEP dx,dy] 'string';
```

SEE ALSO

CHARACTER ROTATE, CHARACTER SCALE, SET CHARACTERS

VAX and IBM FORTRAN GSR

```
CALL PChSca (Name, ScaleX, ScaleY, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
ScaleX, ScaleY are REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PCharSca (    %DESCR Name       : P_VaryingType;
                               ScaleX     : REAL;
                               ScaleY     : REAL;
                        %DESCR AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PCharSca (     CONST  Name       : STRING;
                                ScaleX     : SHORTREAL;
                                ScaleY     : SHORTREAL;
                         CONST  AppliedTo  : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PCharSca( name,scalex,scaley,appliedto )
```

where:

string name,appliedto;
double scalex,scaley;

DESCRIPTION

This routine creates a uniform 2x2 scale matrix to scale the specified characters (Apply/Applied to), where **ScaleX** and **ScaleY** give the scaling factors for the x and y axes.

PS 390 Command and Syntax

```
Name := CHARacter SCAle s [APPLied to name1];
Name := CHARacter SCAle sx,sy [APPLied to name1];
```

SEE ALSO

CHARACTER ROTATE, CHARACTERS, SET CHARACTERS

VAX and IBM FORTRAN GSR

```
CALL PConn (Source, Output, Input, Destination, ErrHnd)
```

where:

Source is a CHARACTER STRING
Output is an INTEGER*4
Input is an INTEGER*4
Destination is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PConnect (   %DESCR Source      : P_VaryingType;
                              Output      : INTEGER;
                              Input       : INTEGER;
                         %DESCR Destination : P_VaryingType;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PConnect (   CONST  Source      : STRING;
                              Output      : INTEGER;
                              Input       : INTEGER;
                         CONST  Destination : STRING;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PConnect( source,output,input,destination)
```

where:

string source,destination;
integer output,input;

DESCRIPTION

This routine connects the **Output** of the function instance **Source** to the **Input** of the function instance or display structure **Destination.**

PS 390 Command and Syntax

```
CONNECT name1<i>:<j>name2;
```

SEE ALSO

DISCONNECT

VAX and IBM FORTRAN GSR

```
CALL PCopyV (Name, CopyFrom, Start, Count, ErrHnd) where:
```

Name is a CHARACTER STRING
CopyFrom is a CHARACTER STRING
Start is an INTEGER*4
Count is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PCopyVec (  %DESCR Name       : P_VaryingType;
                      %DESCR CopyFrom   : P_VaryingType;
                             Start      : INTEGER;
                             Count      : INTEGER;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PCopyVec (  CONST  Name       : STRING;
                      CONST  CopyFrom   : STRING;
                             Start      : INTEGER;
                             Count      : INTEGER;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PCopyVec( name,copyfrom,start,count )
```

where:

string name,copyfrom;
integer start,count;

DESCRIPTION

This routine creates a vector list **Name** containing a group of consecutive vectors copied from another vector list **CopyFrom** or a labels node containing a group of consecutive labels, where **Start** is the first vector to be copied and **Count** is the number of vectors to be copied.

PS 390 Command and Syntax

```
Name := COPY name1 [START=] i [,] [COUNT=] n;
```

VAX and IBM FORTRAN GSR

```
CALL PDefPa (Name, Segments, Pattern, Continuous, Match, Length,
             ErrHnd)
```

where

Name is a CHARACTER STRING
Segments is an INTEGER*4
Pattern is an INTEGER*4 (Segments) Array
Continuous is a LOGICAL
Match is a LOGICAL
Length is a REAL
ErrHnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PDefPatt ( %DESCR Name 1     : P_VaryingType;
                            Segments   : INTEGER;
                        VAR Pattern    : P_PatternType;
                            Continuous : BOOLEAN;
                            Match      : BOOLEAN;
                            Length     : REAL;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PDefPatt (  CONST Name 1     : STRING;
                            Segments   : INTEGER;
                      CONST Pattern    : P_PatternType;
                            Continuous : BOOLEAN;
                            Match      : BOOLEAN;
                            Length     : REAL;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PDefPatt( name1,segments,pattern,continuous,match,length )
```

where:

string name1;
integer segments;
P_PatternType pattern;
boolean continuous,match;
double length;

DESCRIPTION

This routine defines a pattern that can be used to pattern a vector list or curve. **Segments** defines the number of integers used to define the pattern, those integers given by **pattern**. **Continuous** tells whether or not patterning is to go across multiple vectors. **Match** tells if the pattern length is to be adjusted to make the patterning terminate precisely at the endpoints. **Length** gives the pattern length.

PS 390 Command and Syntax

```
Name  := PATtern i [AROUND_CORNERS][MATCH/NOMATCH] LENgth r;
```

VAX and IBM FORTRAN GSR

```
CALL PDelet (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
Errhand is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PDelete (   %DESCR Name   : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PDelete (   CONST Name   : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PDelete( name )
```

where:

string name;

DESCRIPTION

This routine deletes the name of any previously defined data structure **name**. After this routine is issued, all functions and data structures referring to **name** will no longer include the data that was associated with **name**.

PS 390 Command and Syntax

```
DELete name[,name1...namen];
```

SEE ALSO

NIL, FORGET

VAX FORTRAN UTILITY ROUTINE

```
CALL PDelim (Newd, ErrHnd)
```

where:

Newd is a single character CHARACTER STRING that is the new string delimiter
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This routine can be used to change the string delimiting character. The default string delimiter is " (double quote).

VAX and IBM FORTRAN GSR

```
CALL PDeLOD (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PDecLOD (  %DESCR Name      : P_VaryingType;
                     %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PDecLOD (  CONST Name      : STRING;
                     CONST AppliedTo : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PDecLOD( name,appliedto )
```

where:

string name,appliedto;

DESCRIPTION

This routine decrements the current level-of-detail by 1.

PS 390 Command and Syntax

```
Name := DECrement LEVel_of_detail [APPLied to name1];
```

SEE ALSO

INCREMENT_LEVEL_OF_DETAIL, IF_LEVEL_OF_DETAIL

VAX and IBM FORTRAN GSR

```
CALL PDelW (Name, ErrHnd)
```

where

Name is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PDelWild ( %DESCR Name       : P_VaryingType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PDelWild ( CONST Name       : STRING;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PDelWild( name )
string name;
```

DESCRIPTION

This routine deletes all names that begin with the string specified by **name.**

PS 390 Command and Syntax

```
DELete ANY_STRING*;
```

SEE ALSO

NIL, FORGET

VAX and IBM FORTRAN GSR

```
CALL PDi (Source, Output, Input, Destination, ErrHnd)
```

where:

Source is a CHARACTER STRING
Output is an INTEGER*4
Input is an INTEGER*4
Destination is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PDisc (   %DESCR Source      : P_VaryingType;
                           Output      : INTEGER;
                           Input       : INTEGER;
                    %DESCR Destination : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PDisc (    CONST Source      : STRING;
                           Output      : INTEGER;
                           Input       : INTEGER;
                     CONST Destination : STRING;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PDisc( source,output,input,destination)
```

where:

string source,destination;
integer output,input;

DESCRIPTION

This routine disconnects the output number **Output** of the function instance **Source** from the **Input** of the function instance or display structure **Destination.**

PS 390 Command and Syntax

```
DISCONNect name1<i>:<j>name2;
```

SEE ALSO

CONNECT

VAX and IBM FORTRAN GSR

```
PDiAll (Source, ErrHnd)
```

where:

Source is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PDiscAll (   %DESCR Source     : P_VaryingType;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PDiscAll (   CONST Source     : STRING;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PDiscAll( source )
```

where:

string source;

DESCRIPTION

This routine disconnects all outputs of **Source** from all inputs to function instances or display structures.

PS 390 Command and Syntax

```
DISCONNect name1:ALL;
```

SEE ALSO

CONNECT

VAX FORTRAN UTILITY ROUTINE

```
CALL PDInfo (Channel, Device, Status, ErrHnd)
```

where

Channel is an INTEGER*4 that is the VAX Q I/O channel number

Device is an INTEGER*4 that is the device code, where:
1 is unused
2 is the code for asynchronous interface
3 is the code for the parallel interface
4 is the code for the Ethernet interface

Status is an INTEGER*4 that is the status, where;
0 is not attached
1 is attached

ErrHand is the user-defined error-handler subroutine

VAX PASCAL UTILITY ROUTINE

```
[GLOBAL] PROCEDURE PDevInfo (  VAR Channel_num : INTEGER;
                               VAR DEVICE_TYPE : INTEGER;
                               VAR Dev_status  : INTEGER;
                             PROCEDURE Error_Handler (Err : INTEGER));
```

where:

Channel_num is the VAX Q I/O channel number

Device_type is the device code, where:
1 is unused
2 is the code for asynchronous interface
3 is the code for the parallel interface
4 is the code for the Ethernet interface

Dev_status is the status, where;
0 is not attached
1 is attached

DESCRIPTION

This procedure is used to return the Q I/O channel number so that users do not need to detach from the GSRs while doing Physical I/O.

VAX and IBM FORTRAN GSR

```
CALL PDiOut (Source, Output, ErrHnd)
```

where:

Source is a CHARACTER STRING
Output is an INTEGER*4
Errhnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PDiscOut (   %DESCR Source  : P_VaryingType;
                              Output  : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PDiscOut (   CONST Source  : STRING;
                             Output  : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PDiscOut( source,out )
```

where:

string source;
integer out;

DESCRIPTION

This routine disconnects the **Output** of the function instance **Source** from all inputs to function instances or display structures.

PS 390 Command and Syntax

```
DISCONNect name1<i>:ALL;
```

SEE ALSO

CONNECT

VAX and IBM FORTRAN GSR

```
CALL PDisp (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PDisplay (   %DESCR Name       : P_VaryingType;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PDisplay (   CONST Name       : STRING;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PDisplay( name )
```

where:

string name;

DESCRIPTION

This routine displays a data structure **Name**.

PS 390 Command and Syntax

```
DISPlay Name;
```

SEE ALSO

REMOVE

VAX and IBM FORTRAN UTILITY ROUTINE

```
CALL PDtach (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX and IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PDetach (PROCEDURE Error_Handler (Error : INTEGER));
```

UNIX/C UTILITY ROUTINE

```
#include <ps300/gsrext.h>
PDetach()
```

DESCRIPTION

This routine detaches (disconnects) the communications link established between the host and the PS 390. This routine should always be the last GSR routine called by the application program.

SEE ALSO

PAttch, PAttach

VAX and IBM FORTRAN GSR

```
CALL PEnd (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PEnd (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PEnd (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PEnd();
```

DESCRIPTION

This routine is used with the BEGIN routine to group a set of viewing and/or modeling commands so that they appear to be executed simultaneously.

PS 390 Command and Syntax

```
Begin...END;
```

SEE ALSO

BEGIN

VAX and IBM FORTRAN GSR

```
CALL PEndOp (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PEndOpt (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PEndOpt (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PEndOpt();
```

DESCRIPTION

This routine is used with the OPTIMIZE STRUCTURE routine, which places the PS 390 in an "optimization mode" in which certain elements of the display structure are created in a way that minimizes Display Processor traversal time. This routine must be called to complete the sequence. It is strongly suggested that users familiarize themselves with the OPTIMIZE command documentation in the PS 390 Command Summary before using this routine to learn the full ramifications and constraints of this command.

PS 390 Command and Syntax

```
OPTIMIZE STRUCTURE;...END OPTIMIZE;
```

SEE ALSO

OPTIMIZE STRUCTURE

VAX and IBM FORTRAN GSR

```
CALL PEndS (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PEndS (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PEndS (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PEndS();
```

DESCRIPTION

This routine is used with the BEGIN STRUCTURE routine to group a set of viewing and/or modeling commands so that each element does not need to be explicitly named.

PS 390 Command and Syntax

```
Name := BEGIN_Structure...END_Structure;
```

SEE ALSO

BEGIN STRUCTURE

VAX and IBM FORTRAN GSR

```
CALL PEraPa (Name, ErrHnd)
```

where

Name is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PEraPatt ( %DESCR Name1 : P_VaryingType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PEraPatt ( CONST Name1 : STRING;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PEraPatt( name )
```

where:

string name;

DESCRIPTION

This routine removes a pattern from **name** if **name** is a patterned vector list or curve.

PS 390 Command and Syntax

```
ERASE PATTERN FROM Name;
```

SEE ALSO

PATTERN, PATTERN WITH

VAX and IBM FORTRAN GSR

```
CALL PEyeBk (Name, DistBack, DistHoriz, DistVert, Wide, Front, Back,
             Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
DistBack is a REAL*4
DistHoriz is a REAL*4 (positive for right/negative for left)
DistVert is a REAL*4 (positive for up/negative for down)
Wide is a REAL*4
Front is a REAL*4
Back is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PEyeBack (   %DESCR Name      : P_VaryingType;
                              DistBack  : REAL;
                              DistHoriz : REAL;
                              DistVert  : REAL;
                              Wide      : REAL;
                              Front     : REAL;
                              Back      : REAL;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PEyeBack (   CONST  Name      : STRING;
                              DistBack  : SHORTREAL;
                              DistHoriz : SHORTREAL;
                              DistVert  : SHORTREAL;
                              Wide      : SHORTREAL;
                              Front     : SHORTREAL;
                              Back      : SHORTREAL;
                       CONST  AppliedTo : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>
PEyeBack( name,distback,disthoriz,distvert,wide,front,back,appliedto )
```

where:

```
string name;
double distback,disthoriz,distvert,wide,front,back;
string appliedto;
```

DESCRIPTION

This routine specifies a viewing pyramid with the following parametric definitions:

DistBack is the perpendicular distance of the eye from the plane of the viewport.

DistHoriz is the horizontal distance of the eye, right or left from the viewport center (positive for right/negative for left).

DistVert is the vertical distance of the eye, up or down from the viewport center (positive for up/negative for down).

Wide is the width of the viewport.

Front is the front boundary of the frustum of the viewing pyramid.

Back is the back boundary of the frustum of the viewing pyramid.

PS 390 Command and Syntax

```
Name  := EYE BACK z [option1][option2] from screen area w WIDE [FRONT
         Boundary = zmin BACK Boundary = zmax] [APPLied to name1];
```

SEE ALSO

FIELD_OF_VIEW, WINDOW

VAX and IBM FORTRAN GSR

```
CALL PFn (Name, FunctionName, ErrHnd)
```

where:

Name is a CHARACTER STRING
FunctionName is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PFnInst (   %DESCR Name          : P_VaryingType;
                      %DESCR FunctionName  : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PFnInst (   CONST Name           : STRING;
                      CONST FunctionName   : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PFnInst( name,functionname )
```

where:

string name,fcnname;

DESCRIPTION

This routine creates an instance of an intrinsic PS 390 function.

PS 390 Command and Syntax

```
Name := F:FnName;
```

VAX and IBM FORTRAN GSR

```
CALL PFnN (Name, FunctionName, InOuts, ErrHnd)
```

where:

Name is a CHARACTER STRING
FunctionName is a CHARACTER STRING
InOuts is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PFnInstN (   %DESCR Name          : P_VaryingType;
                       %DESCR FunctionName  : P_VaryingType;
                              In_Outs       : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PFnInstN (   CONST Name          : STRING;
                       CONST FunctionName  : STRING;
                             In_Outs       : INTEGER;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PFnInstN( name,functionname,in_outs )
```

where:

string name,functionname;
integer in_outs;

DESCRIPTION

This routine creates an instance of an intrinsic PS 390 function where **InOuts** is the number of respective inputs or outputs of the function. Intrinsic functions that are used by this routine are:

F:CI(n)
F:CIROUTE(n)
F:CONCATXDATA(n)
F:CROUTE(n)
F:DEMUX(n)
F:INPUTE_CHOOSE(n)
F:MCAT_STRING(n)
F:MINNMAX(n)
F:PASSTHRU(n)
F:ROUTE(n)
FROUTEC(n)
F:SYNC(n).

PS 390 Command and Syntax

```
Name := F:FnName(n);
```

VAX and IBM FORTRAN GSR

```
CALL PFoll (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PFoll (   %DESCR Name     : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PFoll (  CONST Name      : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PFoll( name )
```

where:

string name;

DESCRIPTION

This routine follows a named operation node **name** with another operation node. The user must first call this routine, and then IMMEDIATELY call the routine corresponding to the "transformation-or-attribute command."

PS 390 Command and Syntax

```
FOLLOW name WITH option;
```

SEE ALSO

REMOVE FOLLOWER

VAX and IBM FORTRAN GSR

```
CALL PFont (Name, FontName, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
FontName is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PFont (  %DESCR Name      : P_VaryingType;
                   %DESCR FontName  : P_VaryingType;
                   %DESCR AppliedTo : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PFont (  CONST Name      : STRING;
                   CONST FontName  : STRING;
                   CONST AppliedTo : STRING;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PFont(name,fontname,appliedto)
```

where:

string name,fontname,appliedto;

DESCRIPTION

This routine establishes a character font **FontName** as the working font for the specified display structure **Apply/AppliedTo**.

PS 390 Command and Syntax

```
Name := character FONT fontname [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PForg (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PForget (   %DESCR Name    : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PForget (   CONST Name    : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PForget( name )
```

where:

string name;

DESCRIPTION

This routine removes **Name** from the display and from the dictionary of names. **Name** is any previously defined data structure name.

PS 390 Command and Syntax

```
FORget Name;
```

SEE ALSO

DELETE, NIL

VAX and IBM FORTRAN GSR

```
CALL PFov (Name, Angle, Front, Back, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Front is a REAL*4
Back is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PFov (  %DESCR Name      : P_VaryingType;
                         Angle     : REAL;
                         Front     : REAL;
                         Back      : REAL;
                  %DESCR AppliedTo : P_VaryingType;
                PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PFov (   CONST Name      : STRING;
                         Angle     : SHORTREAL;
                         Front     : SHORTREAL;
                         Back      : SHORTREAL;
                   CONST AppliedTo : STRING;
                PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PFov(name,angle,front,back,appliedto)
```

where:

string name,appliedto;
double angle,front,back;

DESCRIPTION

This routine specifies a right rectangular viewing pyramid with the following parametric definitions:

Angle is the angle of view from the eye.

Front is the front boundary of the frustum of the viewing pyramid.

Back is the back boundary of the frustum of the viewing pyramid.

PS 390 Command and Syntax

```
Name  := Field_Of_View Angle FRONT boundary = zmin
         BACK boundary = zmax [APPLied to name1];
```

SEE ALSO

EYEBACK, WINDOW

VAX and IBM FORTRAN UTILITY ROUTINE

```
CALL PGet (String, MessageLength, ErrHnd)
```

where:

String is a CHARACTER STRING that contains the message read from the PS 390
MessageLength is an INTEGER*4 that is the length of String
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL UTILITY ROUTINE

```
PROCEDURE PGet (  %DESCR String : P_VaryBufType;
                  PROCEDURE Error_Handler (Error : INTEGER));
```

NOTE

The parameter **String** must be declared to be a P_VaryBufType.

IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PGet (  VAR String : STRING;
                  PROCEDURE Error_Handler (Error : INTEGER));
```

UNIX/C UTILITY ROUTINE

```
#include <ps300/gsrext.h>

PGet(string,max_len)
```

where:

string string;
integer max_len;

DESCRIPTION

The PGet routine is used to poll the PS 390 for input records by requesting a message that has been sent to the PS 390 function HOST_MESSAGE. The actual message contents are returned in **String**. The number of bytes read are returned in **MessageLength**. If a PGet call is issued and no message exists to be sent back to the host, then the returned length of the message **MessageLength** is 0. Otherwise, the length of the message is greater than 0, and indicates the true number of bytes in the message.

NOTE

If the default value for input <2> or input <3> of HOST_MESSAGEB is changed by the user to be something other than a single carriage return, the above description no longer applies.

VAX and IBM FORTRAN UTILITY ROUTINE

```
CALL PGetW (String, MessageLength, ErrHnd)
```

where:

String is a CHARACTER STRING that contains the message read from the PS 390
MessageLength is an INTEGER*4 that is the length of String
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL UTILITY ROUTINE

```
PROCEDURE PGetWait (  %DESCR String : P_VaryBufType;
                      PROCEDURE Error_Handler (Error : INTEGER));
```

NOTE

The parameter **String** must be declared to be a P_VaryBufType.

IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PGetWait (  VAR String : STRING;
                      PROCEDURE Error_Handler (Error : INTEGER));
```

UNIX/C UTILITY ROUTINE

```
#include <ps300/gsrext.h>

PGetWait(string,max_len)
```

where:

string string;
integer max_len;

DESCRIPTION

The PGetW routine is used to poll the PS 390 for input records by requesting a message that has been sent to the PS 390 function HOST_MESSAGE. If no message exists to be read, the PGetW routine will wait until a message arrives from HOST_MESSAGE. The actual message contents are returned in **String**. The number of bytes read are returned in **MessageLength**.

NOTE

If the default value for input <2> of HOST_MESSAGEB is changed by the user to be something other than a single carriage return, the above description no longer applies.

VAX and IBM FORTRAN GSR

```
CALL PGUCPU (ErrHnd)
```

VAX PASCAL GSR

```
PROCEDURE PGiveUpCPU (PROCEDURE Error_Handler (Error : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PGiveUpCPU (PROCEDURE Error_Handler (Error : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>
PGiveUpCPU()
```

DESCRIPTION

This routine is used to avoid potential timing problems when using the F:ALLOW_VECNORM function for CPK renderings. It causes the command interpreter to terminate execution temporarily and allow other functions to be activated.

VAX and IBM FORTRAN GSR

```
CALL PIfBit (Name, BitNumber, OnOff, Apply, ErrHnd) where:
```

Name is a CHARACTER STRING
BitNumber is an INTEGER*4
OnOff is a LOGICAL*1
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PIfBit (    %DESCR Name       : P_VaryingType;
                             BitNumber  : INTEGER;
                             OnOff      : BOOLEAN;
                      %DESCR AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PIfBit (     CONST Name       : STRING;
                             BitNumber  : INTEGER;
                             OnOff      : BOOLEAN;
                       CONST AppliedTo  : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PIfBit(name,bitnumber,onoff,appliedto)
```

where:

string name,appliedto;
integer bitnumber;
boolean onoff;

DESCRIPTION

This routine refers to a data structure if an attribute bit has a specified setting (On or Off), where **BitNumber** indicates which bit to test and **OnOff** is TRUE for ON and FALSE for OFF.

PS 390 Command and Syntax

```
Name := IF conditional_BIT n is OnOff [THEN name1];
```

SEE ALSO

SET CONDITIONAL_BIT

VAX and IBM FORTRAN GSR

```
CALL PIfLev (Name, Level, Comparison, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Level is an INTEGER*4
*Comparison is an INTEGER*4 corresponding to the comparison test to be performed
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PIfLevel (   %DESCR Name      : P_VaryingType;
                              Level     : INTEGER;
                              Comparison: INTEGER;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PIfLevel (    CONST Name      : STRING;
                              Level     : INTEGER;
                              Comparison: INTEGER;
                        CONST AppliedTo : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PIfLevel(name,level,test,appliedto)
```

where:

string name,appliedto;
integer level,test;

DESCRIPTION

This routine controls the traversal of a display structure based on the result of the relationship (**comparison,test**) between the current level of detail and the specified **level**.

PS 390 Command and Syntax

```
Name := IF LEVEL_of_detail relationship n [THEN name1];
```

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

FORTRAN

Mnemonic	Comparison	INTEGER*4_Value
PCLES	<	0
PCEQL	=	1
PCLEQL	<=	2
PCGTR	>	3
PCNEQL	<>	4
PCGEQL	>=	5

Pascal and UNIX

Mnemonic	Comparison	INTEGER*4_Value
P_LES	<	0
P_EQL	=	1
P_LEQL	<=	2
P_GTR	>	3
P_NEQL	<>	4
P_GEQL	>=	5

SEE ALSO

SET LEVEL_OF_DETAIL

VAX and IBM FORTRAN GSR

```
CALL PIfPha (Name, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Onoff is a LOGICAL*1 defined
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PIfPhase (   %DESCR Name       : P_VaryingType;
                              OnOff      : BOOLEAN;
                       %DESCR AppliedTo  : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PIfPhase (   CONST Name       : STRING;
                             OnOff      : BOOLEAN;
                       CONST AppliedTo  : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PIfPhase( name,onoff,appliedto )
```

where:

string name,appliedto;
boolean onoff;

DESCRIPTION

This routine controls the traversal of a data structure **apply** if the PHASE attribute is in the specified state, ON or OFF. **Onoff** is TRUE for ON and FALSE for OFF. The state of the phase attribute is controlled by the Set Rate and Set Rate External routines.

PS 390 Command and Syntax

```
Name := IF PHASE state [THEN name1];
```

SEE ALSO

SET RATE, SET RATE EXTERNAL

VAX and IBM FORTRAN GSR

```
CALL PIllum (Name, X, Y, Z, Hue, Saturation, Intensity, Ambient,
             ErrHnd)
```

where

Name is a CHARACTER STRING
X is a REAL*4
Y is a REAL*4
Z is a REAL*4
Hue is a REAL*4
Saturation is a REAL*4
Intensity is a REAL*4
Ambient is a REAL*4
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PIllumin ( %DESCR Name       : P_VaryingType;
                            X          : REAL;
                            Y          : REAL;
                            Z          : REAL;
                            Hue        : REAL;
                            Saturation : REAL;
                            Intensity  : REAL;
                            Ambient    : REAL:   {default 1}
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PIllumin (  CONST Name       : STRING;
                            X          : REAL;
                            Y          : REAL;
                            Z          : REAL;
                            Hue        : REAL;
                            Saturation : REAL;
                            Intensity  : REAL;
                            Ambient    : REAL:   {default 1}
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PIllumin (name,x,y,z,hue,saturation,intensity,ambient)
```

where:

```
string name;
double x,y,z,hue,saturation,intensity;
double ambient;
```

DESCRIPTION

This routine defines polygon illumination characteristics used by the rendering firmware in the PS 390 to produce shaded renderings. The direction to the light source is specified by **x, y, z**. The color is specified by **Hue**, **Saturation**, and **Intensity**. Its contribution to ambient lighting is specified by **Ambient** (0 to 1).

PS 390 Command and Syntax

```
Name  := ILLUMINATION X, Y, Z [COLOR h[,s[,i]]] [AMBIENT];
```

VAX and IBM FORTRAN GSR

```
CALL PIncl  (Name1, Name2, ErrHnd)
```

where:

Name1 is a CHARACTER STRING
Name2 is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PIncl (   %DESCR Name1         : P_VaryingType;
                    %DESCR Name2         : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PIncl (    CONST Name1         : STRING;
                     CONST Name2         : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PIncl( name1,name2 )
```

where:

string name1,name2;

DESCRIPTION

This routine is used to include one named data structure **Name1** in a named instance of another data structure **Name2**.

PS 390 Command and Syntax

```
INCLude name1 IN name2;
```

SEE ALSO

REMOVE FROM

VAX and IBM FORTRAN GSR

```
CALL PInit (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PInit (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PInit (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PInit();
```

DESCRIPTION

This routine restores the PS 390 to its initial state. There are no user-defined names, data structures, or function connections; and no data structures are displayed.

PS 390 Command and Syntax

```
INITialize;
```

SEE ALSO

INITIALIZE CONNECTIONS, INITIALIZE DISPLAY, INITIALIZE NAMES

VAX and IBM FORTRAN GSR

```
CALL PInitC (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PInitC (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PInitC (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PInitC();
```

DESCRIPTION

This routine breaks all user-defined function connections.

PS 390 Command and Syntax

```
INITialize CONNections;
```

SEE ALSO

INITIALIZE DISPLAY, INITIALIZE NAMES

VAX and IBM FORTRAN GSR

```
CALL PInitD (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PInitD (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PInitD (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PInitD();
```

DESCRIPTION

This routine removes all display structures from the display list.

PS 390 Command and Syntax

```
INITialize DISPlay;
```

SEE ALSO

INITIALIZE CONNECTIONS, INITIALIZE NAMES

VAX and IBM FORTRAN GSR

```
CALL PInitN (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PInitN (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PInitN (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PInitN();
```

DESCRIPTION

This routine clears the name dictionary of all user-defined structures and function instance names.

PS 390 Command and Syntax

```
INITialize NAMES;
```

SEE ALSO

INITIALIZE CONNECTIONS, INITIALIZE DISPLAY

VAX and IBM FORTRAN GSR

```
CALL PInLOD (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PIncLOD (  %DESCR Name      : P_VaryingType;
                     %DESCR AppliedTo : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PIncLOD (  CONST Name      : STRING;
                     CONST AppliedTo : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PIncLOD(name,appliedto)
```

where:

string name,appliedto;

DESCRIPTION

This routine increments the current level of detail by 1.

PS 390 Command and Syntax

```
Name := INCRement LEVel_of_detail[APPLied to name1];
```

SEE ALSO

DECREMENT LEVEL_OF_DETAIL, IF LEVEL_OF_DETAIL

VAX and IBM FORTRAN GSR

```
CALL PInst (Name1, Name2, ErrHnd)
```

where:

Name1 is a CHARACTER STRING
Name2 is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PInst (   %DESCR Name1       : P_VaryingType;
                    %DESCR Name2       : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER)); DEFINITION
```

IBM PASCAL GSR

```
PROCEDURE PInst (    CONST Name1       : STRING;
                     CONST Name2       : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PInst( name1,name2 )
```

where:

string name1,name2;

DESCRIPTION

This routine creates an instance node **Name1** with pointers to the data structure referenced **Name2**.

PS 390 Command and Syntax

```
Name1:= INSTance of name2[,....,namen];
```

VAX and IBM FORTRAN GSR

```
CALL PLaAdd (X, Y, Z, Label, ErrHnd)
```

where:

X,Y,Z are REAL*4
Label is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PLabAdd (          X          : REAL;
                             Y          : REAL;
                             Z          : REAL;
                   %DESCR Str           : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PLabAdd (          X          : SHORTREAL;
                             Y          : SHORTREAL;
                             Z          : SHORTREAL;
                      CONST Str         : STRING;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PLabAdd( x,y,z,str )
```

where:

double x,y,z;
string str;

DESCRIPTION

This routine is the middle call in creating a label block. It must be called to specify or add a label to a previously opened label. A complete label block requires routines for Begin, Add, and End.

PS 390 Command and Syntax

Together, the above three routines implement the PS 390 command:

```
Name  := LABELS  x, y [,z] 'string'
                    :
                    :
                 xi, yi [,zi] 'string';
```

VAX and IBM FORTRAN GSR

```
CALL PLaBeg (LabelBlock, StepX, StepY, ErrHnd)
```

where:

LabelBlock is a CHARACTER STRING
StepX and StepY are REAL*4
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PLabBegn (   %DESCR LabelBlock: P_VaryingType;
                              StepX     : REAL;
                              StepY     : REAL;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PLabBegn (    CONST LabelBlock: STRING;
                              StepX     : SHORTREAL;
                              StepY     : SHORTREAL;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PLabBegn( labelblock,stepx,stepy )
```

where:

string labelblock
double stepx,stepy;

DESCRIPTION

This routine must be called to create and open a label block. A complete label block requires routines for Begin, Add, and End.

NOTE

The **stepx** and **stepy** parameters allow the steps between the label blocks to be specified in terms of x and y. If stepx and stepy were specified as 1.0 and 0.0 respectively, each successive character would be displayed one unit to the right of and horizontally aligned with the preceding character. This applies to all labels within the label block. It should prove useful for those users who wish to make vertical or slanted label blocks. Users cannot send to <step> of a label block; a message from the CI results.

PS 390 Command and Syntax

Together, the three routines implement the PS 390 command:

```
Name  := LABELS  x, y [,z] 'string'
                    :
                    :
                 xi, yi [,zi] 'string';
```

VAX and IBM FORTRAN GSR

```
CALL PLaEnd (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PLabEnd (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PLabEnd (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PLabEnd();
```

DESCRIPTION

This routine must be called to complete the creation of a label block. A complete label block requires routines for Begin, Add, and End.

PS 390 Command and Syntax

Together, the above three routines implement the PS 390 command:

```
Name  := LABELS  x, y [,z] 'string'
                    :
                    :
                 xi, yi [,zi] 'string';
```

UNIX/C UTILITY ROUTINE

```
#include <ps300/gsrext.h>

PLoad( muxbyte,buf,buflen )

    integer muxbyte,buflen;
    char *buf;
```

DESCRIPTION

PLoad sends the data in **buf** to the current GSR library output device. It prefixes the data in **buf** with the **muxbyte** and four more bytes of output device-dependent preamble before sending it. Therefore, the application program should only store binary data starting at **buf**[5]. **Buflen** is the size of **buf** in bytes. The value of the mux byte (also called the routing byte) determines where the PS 390 CIROUTE function will route the data sent along with it. Refer to *Section RM7 Host Input Data Flow* for complete definitions of routing byte values.

SEE ALSO

PSavBeg, PSavEnd

VAX and IBM FORTRAN GSR

```
CALL PLookA (Name, At, From, Up, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
At is defined as REAL*4 At(3)
From is defined as REAL*4 From(3)
Up is defined as REAL*4 Up(3)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PLookAt (   %DESCR Name      : P_VaryingType;
                         VAR At        : P_VectorType;
                         VAR From      : P_VectorType;
                         VAR Up        : P_VectorType;
                      %DESCR AppliedTo : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PLookAt (    CONST Name      : STRING;
                       CONST At        : P_VectorType;
                       CONST From      : P_VectorType;
                       CONST Up        : P_VectorType;
                       CONST AppliedTo : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PLookAt(name,at,from,up,appliedto)
```

where:

string name,appliedto;
P_VectorType at,from,up;

DESCRIPTION

This routine, when used with WINDOW, EYE BACK, or FIELD_OF_VIEW routines, fully specifies the portion of the data space that will be viewed as well as the viewer's orientation in data space. It has the following parametric definitions:

At is the point being looked at in data space coordinates.
From is the location of the viewer's eye in data space coordinates.
Up indicates the screen "up" direction.

PS 390 Command and Syntax

```
Name := LOOK AT ax,ay,az FROM fx,fy,fz [UP ux,uy,uz] [APPLied to name1];

Name := LOOK FROM fx,fy,fz AT ax,ay,az [UP ux,uy,uz] [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PMat22 (Name, Mat, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Mat is the matrix to be sent and is defined: REAL*4 Mat(4,4)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PMat2x2 (   %DESCR Name      : P_VaryingType;
                         VAR Mat       : P_MatrixType;
                      %DESCR AppliedTo : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PMat2x2 (    CONST Name      : STRING;
                       CONST Mat       : P_MatrixType;
                       CONST AppliedTo : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PMat2x2( name,matrix,appliedto )
```

where:

string name,appliedto;
P_MatrixType matrix;

DESCRIPTION

This routine creates a special 2x2 transformation matrix that applies to characters in the data structure that follows **Apply/AppliedTo**.

PS 390 Command and Syntax

```
Name := Matrix_2x2 m11, m12,
                   m21, m22, [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PMat33 (Name, Mat, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Mat is the matrix to be sent and is defined: REAL*4 Mat(4,4)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PMat3x3 (   %DESCR Name      : P_VaryingType;
                         VAR Mat       : P_MatrixType;
                      %DESCR AppliedTo : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PMat3x3 (    CONST Name      : STRING;
                       CONST Mat       : P_MatrixType;
                       CONST AppliedTo : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PMat3x3( name,matrix,appliedto )
```

where:

string name,appliedto;
P_MatrixType matrix;

DESCRIPTION

This routine creates a special 3x3 transformation matrix that applies to the specified data (vector lists and/or characters) that follow **Apply/AppliedTo**.

PS 390 Command and Syntax

```
Name := Matrix_3x3 m11, m12, m13
                   m21, m22, m23
                   m31, m32, m33 [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PMat43 (Name, Mat, Vec, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Mat is the matrix to be sent and is defined: REAL*4 Mat(4,4)
Vec is the x,y,z translation to be sent and is defined: REAL*4 Vec(3)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PMat4x3 (   %DESCR Name      : P_VaryingType;
                         VAR Mat       : P_MatrixType;
                         VAR Vec       : P_VectorType
                      %DESCR AppliedTo : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PMat4x3 (    CONST Name      : STRING;
                       CONST Mat       : P_MatrixType;
                       CONST Vec       : P_VectorType
                       CONST AppliedTo : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PMat4x3(name,matrix,vec,appliedto)
```

where:

string name,appliedto;
P_MatrixType matrix;
P_VectorType vec;

DESCRIPTION

This routine creates a special 4x3 matrix that applies to the specified data (vector lists and/or characters) that follow **Apply/Applied to**.

PS 390 Command and Syntax

```
Name := Matrix_4x3 m11, m12, m13
                   m21, m22, m23
                   m31, m32, m33
                   m41, m42, m43 [APPLied to name1];
```

NOTE

The matrix_4x3 command is sent in two parts:
1) a 3x3 matrix sent in **Mat**.
2) a 3D-translation vector (4th row) sent in **Vec**.

VAX and IBM FORTRAN GSR

```
CALL PMat44 (Name, Mat, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Mat is the matrix to be sent and is defined: REAL*4 Mat(4,4)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PMat4x4 (   %DESCR Name      : P_VaryingType;
                         VAR Mat       : P_MatrixType;
                      %DESCR AppliedTo : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PMat4x4 (    CONST Name      : STRING;
                       CONST Mat       : P_MatrixType;
                       CONST AppliedTo : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>
```

```
PMat4x4(name,matrix,appliedto)
```

where:

string name,appliedto;
P_MatrixType matrix;

DESCRIPTION

This routine creates a special 4x4 matrix that applies to the specified data (vector lists and/or characters) that follow **Apply/AppliedTo**.

PS 390 Command and Syntax

```
Name := Matrix_4x4 m11, m12, m13, m14
                   m21, m22, m23, m24
                   m31, m32, m33, m34
                   m41, m42, m43, m44 [APPLied to name1];
```

VAX and IBM FORTRAN UTILITY PROCEDURE

```
CALL PMuxCI (CIchan, ErrHnd)
```

where:

CIchan is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL UTILITY ROUTINE

```
PROCEDURE PMuxCI (  NewCIChan : INTEGER;
                    PROCEDURE Error_Handler (Error : INTEGER));
```

IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PMuxCI (  NewCIChan : INTEGER;
                    PROCEDURE Error_Handler (Error : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PMuxCI(new_ci_chan)
```

where:

integer new_ci_chan;

DESCRIPTION

This routine redefines the CIROUTE output channel accessed as the Binary CI channel. The standard and default CI channel is 2. This routine is provided for the implementation of multiple command interpreters.

VAX and IBM FORTRAN UTILITY PROCEDURE

```
CALL PMuxG (MuxChn, ErrHnd)
```

where:

MuxChn is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL UTILITY ROUTINE

```
PROCEDURE PMuxG (  NewMuxChan : INTEGER;
                   PROCEDURE Error_Handler (Error : INTEGER));
```

IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PMuxG (  NewMuxChan : INTEGER;
                   PROCEDURE Error_Handler (Error : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PMuxG(new_mux_chan)
```

where:

integer new_mux_chan;

DESCRIPTION

The routine redefines the CIROUTE output channel being currently accessed as the "generic" channel by PPutG and PPutGX. The call is provided to support the future implementation of custom user-functions connected to various outputs of CIROUTE.

MuxChn = 1: Send to parser. CIROUTE<3>
MuxChn = 2: Send to READSTREAM CIROUTE<4> etc.

VAX and IBM FORTRAN UTILITY ROUTINE

```
CALL PMuxP (PrsChn, ErrHnd)
```

where:

PrsChn is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL UTILITY ROUTINE

```
PROCEDURE PMuxPars (  NewParseChan : INTEGER;
                      PROCEDURE Error_Handler (Error : INTEGER));
```

IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PMuxPars (  NewParseChan : INTEGER;
                      PROCEDURE Error_Handler (Error : INTEGER));
```

UNIX/C UTILITY ROUTINE

```
#include <ps300/gsrext.h>

PMuxPars(new_parse_chan)
```

where:

integer new_parse_chan;

DESCRIPTION

This routine redefines the CIROUTE output channel accessed by PPutP(ars). The call allows for the implementation and support of multiple Parsers. The standard and default Parser channel is 1.

VAX and IBM FORTRAN GSR

```
CALL PNil (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PNameNil (   %DESCR Name       : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PNameNil (   CONST Name       : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PNameNil( name )
```

where:

string name;

DESCRIPTION

This routine names a null data structure. When this routine is used to redefine **Name**, **Name** is kept in the name dictionary but any data structures previously associated with it are removed. **Forget** does just the opposite of **Nil**.

PS 390 Command and Syntax

```
Name := NIL;
```

SEE ALSO

DELETE, FORGET

VAX and IBM FORTRAN GSR

```
CALL POpt (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE POptStru (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE POptStru (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

POptStru();
```

DESCRIPTION

This routine is used with the End Optimize routine. When it is called, it places the PS 390 in an "optimization mode" in which certain elements of the display structure are created in a way that minimizes Display Processor traversal time. The End Optimize routine must be called to complete the sequence. It is strongly suggested that users familiarize themselves with the OPTIMIZE command documentation in the PS 390 Command Summary before using this routine to learn the full ramifications and constraints of this command.

PS 390 Command and Syntax

```
OPTIMIZE STRUCTURE;...END OPTIMIZE;
```

SEE ALSO

END OPTIMIZE

VAX and IBM FORTRAN GSR

```
CALL PPatWi (Name, PatternName, ErrHnd)
```

where

Name is a CHARACTER STRING
PatternName is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PPatWith ( %DESCR Name1        : P_VaryingType;
                     %DESCR PatternName : P_VaryingType;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PPatWith (  CONST Name1        : STRING;
                      CONST PatternName : STRING;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PPatWith( curvename,patternname )
```

where:

string curvename,patternname;

DESCRIPTION

This routine patterns the curve of the vector list called **Name** (**curvename**) with the pattern **PatternName**, where PatternName has been defined with a call to the (define) PATTERN routine.

PS 390 Command and Syntax

```
PATTERN Name WITH PatternName;
```

SEE ALSO

PATTERN

VAX and IBM FORTRAN GSR

```
CALL PPlygA (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PPlygAtr ( %DESCR Name : P_VaryingType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PPlygAtr (  CONST Name : STRING;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PPlygAtr(name)
```

where:

string name;

DESCRIPTION

This routine specifies that the attributes named by **Name** and specified in a call to the ATTRIBUTES routine apply to all subsequent polygons until superseded by another call to this routine. This routine is one of five routines used to implement the PS 390 polygon command.

PS 390 Command and Syntax

```
Name  := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [Coplanar]
         POLYGon vertex ... vertex;
```

where vertex is defined as:

```
[S] x,y,z [N x,y,z] [C h[,s[,i]]]
```

SEE ALSO

ATTRIBUTES

VAX and IBM FORTRAN GSR

```
CALL PPlygB (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PPlygBeg ( %DESCR Name  : P_VaryingType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PPlygBeg (  CONST Name  : STRING;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PPlygBeg( name )
```

where:

string name;

DESCRIPTION

This routine begins a polygon display list. The parameter **Name** specifies the name to be given to the polygon display list defined by calls to Polygon Attributes, Polygon Outline, and Polygon List routines (or alternatively Polygon RGB List or Polygon HSI List). This routine is one of five used to implement the PS 390 POLYGON command.

A sequence of 3 to 5 routines must be called to create a polygon list.

Polygon (Begin):
This routine is called to begin the creation of a polygon list.

Polygon (Attributes):
This is an optional routine called to specify the attributes to be applied to the polygon. This routine may be called multiple times.

Polygon (Outline):
This is an optional routine called to specify the intensity or color of the polygon. This routine may be called multiple times.

Polygon (List, RGBList, HSIList):
This routine specifies the vertices of each polygon in the polygon list. These routines may be called multiple times.

Polygon (End):
This routine closes the polygon list.

PS 390 Command and Syntax

```
Name  := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [Coplanar]
         POLYGON vertex ... vertex;
```

where vertex is defined as:

```
[S] x,y,z [N x,y,z] [C h[,s[,i]]]
```

VAX and IBM FORTRAN GSR

```
CALL PPlygE (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PPlygEnd  (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PPlygEnd  (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PPlygEnd();
```

DESCRIPTION

This routine ends the definition of a polygon display list. This routine is one of from three to five routines required to implement the PS 390 POLYGON command.

PS 390 Command and Syntax

```
Name  := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [Coplanar]
         POLYGON vertex ... vertex;
```

where vertex is defined as:

```
[S] x,y,z [N x,y,z] [C h[,s[,i]]]
```

VAX and IBM FORTRAN GSR

```
CALL PPlygH (Coplanar, Nvertices, Vertices, Vedges, Normalspec,
             Normals, ColorSpec, Colors, ErrHnd)
```

where:

Coplanar is a LOGICAL
Nvertices is an INTEGER*4
Vertices is a REAL*4 (4,Nvertices)
Vedges is a LOGICAL*1 (Nvertices)
Normalspec is a LOGICAL
Normals is a REAL*4 (4,Nvertices)
ColorSpec is a LOGICAL
Colors is a REAL*4 (4,Nvertices)
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PPlygHSI  (      Coplanar  : BOOLEAN;
                           NVertices : INTEGER:
                      VAR Vertices   : P_VectorListType;
                           Normalspec  : BOOLEAN;
                      VAR Normals    : P_VectorListType;
                           Colorspec : BOOLEAN;
                      VAR Colors     : P_VectorListType;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PPlygHSI  (        Coplanar  : BOOLEAN;
                             NVertices : INTEGER:
                      CONST Vertices   : P_VectorListType;
                             Normalspec  : BOOLEAN;
                      CONST Normals    : P_VectorListType;
                             Colorspec : BOOLEAN;
                      CONST Colors     : P_VectorListType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PPlygLisHSI(coplanar,nvertices,vertices,normalspec,normals,colorspec,
            colors)
```

where:

```
boolean  coplanar,normspec,colorspec;
integer nvertices;
PVectorListType vertices,normals,colors;
```

DESCRIPTION

This routine defines a polygon within the polygon display list currently being constructed. The routine may be called many times to specify additional polygons for the polygon display currently under construction as named by the Polygon Begin routine. It has the following parametric definitions:

Coplanar determines whether the polygon is coplanar with the previous polygon or not.

.TRUE. = coplanar, .FALSE. = not coplanar

Nvertices specifies the number of vertices in the polygon
Vertices specifies the x, y, and z vertices of the polygon

Vedges specifies the "soft" versus "hard" nature of each edge specified by Vertices.

Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

NormalSpec specifies if the normals to the vectors defining the polygon are specified.

NormalSpec = .TRUE. if specified, NormalSpec = .FALSE. if not specified.

Normals specifies a normal to correspond to each vertex. This parameter is of the same form as: **Vertices**.

ColorSpec specifies if the colors attached to the polygon vertices are specified.

ColorSpec = .TRUE. if specified, ColorSpec = .FALSE. if not specified.

Colors specifies the colors of the vertices of the polygon. It is of the same form as Vertices for FORTRAN programmers:

Colors(1,n) = Hue n
Colors(2,n) = Saturation n
Colors(3,n) = Intensity n

For Pascal and UNIX/C programmers, **Colors** is of the same form as vertices, where:

Colors[n].Draw - Not used
Colors[n].V4[1] = Hue value mapped to a range 0–360.0;
Colors[n].V4[2] = Saturation value mapped to range 0–1;
Colors[n].V4[3] = Intensity value mapped to a range 0–1;

Saturation and intensity values are clamped to the nearest range without warning.

NOTE

In the UNIX/C and Pascal GSRs, **Vedges** is not a separately specified parameter; however **Vertices** (and **Normals**) have the following parametric definitions:

Vertices [n].Draw = False defines the edge as 'soft'
Vertices [n].Draw = True defines the edge as 'hard'
Vertices [n].V4[1] = vertex n: x-coordinate;
Vertices [n].V4[2] = vertex n: y-coordinate;
Vertices [n].V4[3] = vertex n: z-coordinate;

This routine is one of five required to implement the PS 390 POLYGON command.

PS 390 Command and Syntax

```
Name  := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [Coplanar]
         POLYGON vertex ... vertex;
```

where vertex is defined as:

```
[S] x,y,z [N x,y,z] [C h[,s[,i]]]
```

VAX and IBM FORTRAN GSR

```
CALL PPlygL (Coplanar, Nvertices, Vertices, Vedges, Normalspec,
             Normals, ErrHnd)
```

where:

Coplanar is a LOGICAL
Nvertices is an INTEGER*4
Vertices is a REAL*4 (4, Nvertices)
Vedges is a LOGICAL*1 (Nvertices)
Normalspec is a LOGICAL
Normals is a REAL*4 (4, Nvertices)
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PPlygLis  (      Coplanar   : BOOLEAN;
                           NVertices  : INTEGER:
                     VAR Vertices     : P_VectorListType;
                           Normalspec : BOOLEAN;
                     VAR Normals      : P_VectorListType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PPlygLis  (        Coplanar   : BOOLEAN;
                             NVertices  : INTEGER:
                     CONST Vertices     : P_VectorListType;
                             Normalspec : BOOLEAN;
                     CONST Normals      : P_VectorListType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PPlygLis(coplanar,nvertices,vertices,normspec,normals)
```

where:

boolean coplanar,normspec;
integer nvertices;
PVectorListType vertices,normals;

DESCRIPTION

This routine defines a polygon within the polygon display list currently being constructed. The routine may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygB routine call. It has the following parametric definitions:

Coplanar determines whether the polygon is coplanar with the previous polygon or not.

.TRUE. = coplanar, .FALSE. = not coplanar

Nvertices specifies the number of vertices in the polygon

Vertices specifies the x, y, and z vertices of the polygon

Vedges specifies the "soft" versus "hard" nature of each edge specified by Vertices.

Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

NormalSpec specifies if the normals to the vectors defining the polygon are specified.

NormalSpec = .TRUE. if specified, NormalSpec = .FALSE. if not specified.

Normals specifies a normal to correspond to each vertex. This parameter is of the same form as: **Vertices**.

NOTE

In the UNIX/C and Pascal GSRs, **Vedges** is not a separately specified parameter; however **Vertices** has the following parametric definitions:

Vertices [n].Draw = False defines the edge as 'soft'
Vertices [n].Draw = True defines the edge as 'hard'
Vertices [n].V4[1] = vertex n: x-coordinate;
Vertices [n].V4[2] = vertex n: y-coordinate;
Vertices [n].V4[3] = vertex n: z-coordinate;

This routine is one of five required to implement the PS 390 POLYGON command.

PS 390 Command and Syntax

```
Name  := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [Coplanar]
         POLYGON vertex ... vertex;
```

where vertex is defined as:

```
[S] x,y,z [N x,y,z] [C h[,s[,i]]]
```

VAX and IBM FORTRAN GSR

```
CALL PPlygO (Outline, ErrHnd)
```

where

Outline is a REAL*4
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PPlygOtl ( %DESCR Outline : REAL;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PPlygOtl ( CONST Outline : REAL;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PPlygOtl(outline);
```

where:

double outline;

DESCRIPTION

This routine specifies that **Outline** be used as the color (if between 1 and 360) or intensity (if between 0 and 1) of all polygons edges until superseded by another call to the Polygon Outline routine.

PS 390 Command and Syntax

This routine is one of five used to implement the PS 390 POLYGON command:

```
Name  := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [Coplanar]
         POLYGON vertex ... vertex;
```

where vertex is defined as:

```
[S] x,y,z [N x,y,z] [C h[,s[,i]]]
```

VAX and IBM FORTRAN GSR

```
CALL PPlygR (Coplanar, Nvertices, Vertices, Vedges, Normalspec,
             Normals, ColorSpec, RGBVal, ErrHnd)
```

where:

Coplanar is a LOGICAL
Nvertices is an INTEGER*4
Vertices is a REAL*4 (4, Nvertices)
Vedges is a LOGICAL*1 (Nvertices)
Normalspec is a LOGICAL
Normals is a REAL*4 (4, Nvertices)
ColorSpec is a LOGICAL
RGBVal is a REAL*4 (3,Nvertices)
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PPlygRGB  (      Coplanar   : BOOLEAN;
                           NVertices  : INTEGER:
                     VAR Vertices     : P_VectorListType;
                           Normalspec : BOOLEAN;
                     VAR Normals      : P_VectorListType;
                           Colorspec  : BOOLEAN;
                     VAR RGBList      : P_PolyColorType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PPlygRGB  (        Coplanar   : BOOLEAN;
                             NVertices  : INTEGER:
                     CONST Vertices     : P_VectorListType;
                             Normalspec : BOOLEAN;
                     CONST Normals      : P_VectorListType;
                             Colorspec  : BOOLEAN;
                     CONST RGBList      : P_PolyColorType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PPlygLisRGB(coplanar,nvertices,vertices,normalspec,normals,colorspec,
            rgblist)
```

where:

```
boolean  coplanar,normalspec,colorspec;
integer nvertices;
PVectorListType vertices,normals;
PPolyColorType rgblist;
```

DESCRIPTION

This routine defines a polygon within the polygon display list currently being constructed. The routine may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygB routine call. It has the following parametric definitions:

Coplanar determines whether the polygon is coplanar with the previous polygon or not.

.TRUE. = coplanar, .FALSE. = not coplanar

Nvertices specifies the number of vertices in the polygon

Vertices specifies the x, y, and z vertices of the polygon

Vedges specifies the "soft" versus "hard" nature of each edge specified by Vertices.

Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

NormalSpec specifies if the normals to the vectors defining the polygon are specified.

NormalSpec = .TRUE. if specified, NormalSpec = .FALSE. if not specified.

Normals specifies a normal to correspond to each vertex. This parameter is of the same form as: **Vertices**.

ColorSpec specifies if the colors attached to the polygon vertices are specified.

ColorSpec = .TRUE. if specified, ColorSpec = .FALSE. if not specified.

RGBVal specifies the colors of the vertices of the polygon. It is of the same form as Vertices for FORTRAN programmers:

Colors(1,n) = Red intensity n (range 0..255)
Colors(2,n) = Green intensity n (range 0..255)
Colors(3,n) = Blue intensity n (range 0..255)

Out-of-range values are converted to the nearest in-range value without warning.

For Pascal and UNIX/C programmers, **RGBList** specifies the colors associated with the polygon vertices, where:

RGBList[1,n] = Red
RGBList[2,n] = Green
RGBList[3,n] = Blue

P_PolycolorType is defined as:
P_PolycolorType = ARRAY [1..3, 1..P_MaxpolygonSize] OF INTEGER;

All Red, Green, Blue values are mapped to the range 0–255. Out-of-range values are clamped to the nearest in-range value without warning.

NOTE

In the UNIX/C and Pascal GSRs, **Vedges** is not a separately specified parameter; however **Vertices** have the following parametric definitions:

Vertices [n].Draw = False defines the edge as 'soft'
Vertices [n].Draw = True defines the edge as 'hard'
Vertices [n].V4[1] = vertex n: x-coordinate;
Vertices [n].V4[2] = vertex n: y-coordinate;
Vertices [n].V4[3] = vertex n: z-coordinate;

This routine is one of five required to implement the PS 390 POLYGON command.

PS 390 Command and Syntax

```
Name  := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [Coplanar]
         POLYGON vertex ... vertex;
```

where vertex is defined as:

```
[S] x,y,z [N x,y,z] [C h[,s[,i]]]
```

VAX and IBM FORTRAN GSR

```
CALL PPoly  (Name, Order, Dimension, Coeffs, Chords, ErrHnd)
```

where:

Name is a CHARACTER STRING
Order is an INTEGER*4
Dimension is an INTEGER*4
Coeffs is defined: REAL*4 (4, Order+1)
Chords is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PPoly (   %DESCR Name       : P_VaryingType;
                           Order      : INTEGER;
                           Dimension  : INTEGER;
                       VAR Coeffs     : P_VectorListType;
                           Chords     : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PPoly (    CONST Name       : STRING;
                           Order      : INTEGER;
                           Dimension  : INTEGER;
                       VAR Coeffs     : P_VectorListType;
                           Chords     : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PPoly( name,order,dimension,coeffs,chords )
```

where:

string name;
integer order,dimension,chords;
P_VectorListType coeffs;

DESCRIPTION

This routine allows the parametric description of many curve forms without the need to specify or transfer the coordinates of each constituent vector. It has the following parametric definitions:

Order is the order of the polynomial.
Dimension is either 2 or 3 (for 2 or 3 dimensions respectively).
Coeffs represent the x,y,z components of the curve. For UNIX/C and VAX Pascal the parameter takes the following form.

```
Coeffs [i].V4 [1]:= x(order -i+1)
Coeffs [i].V4 [2]:= y(order -i+1)
Coeffs [i].V4 [3]:= z(order -i+1)
Coeffs [i].V4 [4] is not used
```

To further clarify the description:

Coeffs [1].V4 [1] := the coefficient that will be applied to the t^{order} term.
Coeffs [2].V4 [1] := the coefficient that will be applied to the $t^{order-1}$ term in the resultant x(t) function computed by this command.
:
:
etc.

Chords is the number of vectors to be created.

NOTE

The definition for Coeffs in IBM Pascal takes the form *Coeffs(.i.).V4(.1.):=x(order–i+1)*. VAX FORTRAN and IBM FORTRAN take the form *Coeffs(1,i) = x(order–i+1)*. The description of the parameter is otherwise identical for each of the programming languages.

PS 390 Command and Syntax

```
Name  := POLYNOMIAL
         ORDER = Order COEFFICIENTS  = X(i),   Y(i),   Z(i)
                                       X(i-1), Y(i-1), Z(i-1)
                                       :       :       :
                                       X(0),   Y(0),   Z(0)
         CHORDS = Chords;
```

VAX and IBM FORTRAN GSR

```
CALL PPref (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PPref (   %DESCR Name       : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PPref (   CONST Name       : STRING;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PPref(name)
```

where:

string name;

DESCRIPTION

This routine prefixes a named data structure **Name** with an operation node. To prefix something, the user must first call this routine and then IMMEDIATELY call the routine corresponding to the PS 390 "transformation-or-attribute" command.

PS 390 Command and Syntax

```
PREfix Name WITH operation_command;
```

VAX and IBM FORTRAN UTILITY ROUTINE

```
CALL PPurge (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL UTILITY ROUTINE

```
PROCEDURE PPurge (PROCEDURE Error_Handler (Error : INTEGER));
```

IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PPurge (PROCEDURE Error_Handler (Error : INTEGER));
```

UNIX/C UTILITY ROUTINE

```
#include <ps300/gsrext.h>

PPurge();
```

DESCRIPTION

The GSRs always buffer the output to the PS 390 to achieve maximum I/O efficiency. This routine explicitly flushes the output buffer and sends any buffered data to the PS 390.

VAX and IBM FORTRAN UTILITY ROUTINE

```
CALL PPutG (String, Length, ErrHnd)
```

where:

String is a CHARACTER STRING
Length is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

IBM NOTE

No translation from EBCDIC to ASCII is performed by the routine. If translation is required, then the PPutGX routine should be used.

VAX PASCAL UTILITY ROUTINE

```
PROCEDURE PPutG (  %DESCR String : P_VaryingType;
                   PROCEDURE Error_Handler (Error : INTEGER));
```

IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PPutG (  CONST String : STRING; (* send generic *)
                   PROCEDURE Error_Handler (Error : INTEGER));
```

UNIX/C UTILITY ROUTINE

```
#include <ps300/gsrext.h>

PPutG(buffer,length)
```

where:

string buffer;
integer length;

DESCRIPTION

This routine sends the bytes specified in the buffer **String** to the current generic demultiplexing channel of CIROUTE established by PMuxG. **Length** defines the number of bytes to send.

IBM FORTRAN UTILITY ROUTINE

```
CALL PPutGX (String, Length, ErrHnd)
```

where:

String is a CHARACTER STRING
Length is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PPutGX (  CONST Str : STRING; (* send generic xlat *)
                  PROCEDURE Error_Handler (Error : INTEGER));
```

DESCRIPTION

PPutGX: (Put Generic) and translate. This routine sends the bytes specified in the buffer **String** to the current generic demultiplexing channel of CIROUTE established by PMuxG. If translation is not required, then the PPutG routine should be used.

VAX and IBM FORTRAN UTILITY ROUTINE

```
CALL PPutP (String, Length, ErrHnd)
```

where:

String is a CHARACTER STRING
Length is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL UTILITY ROUTINE

```
PROCEDURE PPutPars (  %DESCR Str : P_VaryingType;
                      PROCEDURE Error_Handler (Error : INTEGER));
```

IBM PASCAL UTILITY ROUTINE

```
PROCEDURE PPutPars (  CONST Str : STRING;  (* SEND ASCII *)
                      PROCEDURE Error_Handler (Error : INTEGER));
```

UNIX/c UTILITY ROUTINE

```
#include <ps300/gsrext.h>

PPutPars(string)
```

where:

string string;

DESCRIPTION

This routine sends the characters specified in the buffer **String** to the PS 390 Parser.

VAX and IBM FORTRAN GSR

```
CALL PRasCP (x, y, ErrHnd)
```

where:

x is an INTEGER*4
y is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PRasCp (      x  : INTEGER;
              y  : INTEGER;
        PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PRasCp (      x  : INTEGER;
              y  : INTEGER;
        PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRasCp(x,y);
```

where:

integer x,y;

DESCRIPTION

This routine establishes the current pixel location relative to the current logical device coordinates. **x** and **y** specify the x,y coordinates of the current pixel and must be greater than or equal to 0. The lower-left corner of the logical device coordinates is given by (0,0).

PS 390 Command and Syntax

There is no ASCII command corresponding to PRasCP.

SEE ALSO

Load Pixel Data,
Set Logical Device Coordinates

VAX and IBM FORTRAN GSR

```
CALL PRasER (Color, ErrHnd)
```

where:

Color is an INTEGER*4 (3)
ErrHnd is the user-defined error-handler subroutine

Color(1) is the red index
Color(2) is the green index
Color(3) is the blue index

VAX PASCAL GSR

```
PROCEDURE PRasEr (  COLOR  : P_ColorType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

Color.red is the red index
Color.green is the green index
Color.blue is the blue index

IBM PASCAL GSR

```
PROCEDURE PRasEr (  COLOR  : P_ColorType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

Color.red is the red index
Color.green is the green index
Color.blue is the blue index

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRasEr(color)
```

where:

P_ColorType color;

Color.red is the red index
Color.green is the green index
Color.blue is the blue index

DESCRIPTION

This routine is used in WRPIX mode to erase the entire screen to the **color** specified, which is a set of three indexes into the color lookup table (LUT), one each for red, green, and blue. The LUT contains the actual values used for display.

PS 390 Command and Syntax

There is no ASCII command corresponding to PRasER.

VAX and IBM FORTRAN GSR

```
CALL PRasLd (Xmin, Ymin, Xmax, Ymax, ErrHnd)
```

where:

Xmin is an INTEGER*4
Ymin is an INTEGER*4
Xmax is an INTEGER*4
Ymax is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PRasLd (   Xmin : INTEGER;
                     Ymin : INTEGER;
                     Xmax : INTEGER;
                     Ymax : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRasLd (   Xmin : INTEGER;
                     Ymin : INTEGER;
                     Xmax : INTEGER;
                     Ymax : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRasLd(xmin,ymin,xmax,ymax)
```

where:

integer xmin,ymin,xmax,ymax;

DESCRIPTION

This routine sets the logical device coordinates that are used to position the picture in virtual address space. The PS 390 has a virtual pixel address space from -32768 to 2047 in both x and y. The portion of this space that is actually displayed is from 0 to 1023 in x and from 0 to 863 in y. This routine can be used to reposition an image in screen space without recalculation and only retransmission of the data.

PS 390 Command and Syntax

There is no ASCII command corresponding to PRasLD.

SEE ALSO

Load Pixel Data,
Set Current Pixel Location

VAX and IBM FORTRAN GSR

```
CALL PRasWP (Number, Pixval, ErrHnd)
```

where:

Number is an INTEGER*4
Pixval is an INTEGER*4 (4,Num) Array
ErrHnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PRasWP (        Number : INTEGER;
                      VAR Pixval : P_RunClrArrayType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRasWP (        Number : INTEGER;
                      VAR Pixval : P_RunClrArrayType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRasWP(number,pixval)
```

where:

integer number;
P_RunColorType pixval[];

DESCRIPTION

This routine loads the current pixel location with the pixel values. **Number** specifies the number of entries in Pixval array. Each value in **Pixval** has the following structure:

FORTRAN		UNIX/C or Pascal
Pixval (x,1)	or	**Pixval[x].count** is the repetition count
Pixval (x,2)	or	**Pixval[x].red** is the red index
Pixval (x,3)	or	**Pixval[x].green** is the green index
Pixval (x,4)	or	**Pixval[x].blue** is the blue index.

PS 390 Command and Syntax

There is no ASCII command corresponding to PRASWP.

SEE ALSO

Set Current Pixel Location,
Set Logical Device Coordinates

VAX and IBM FORTRAN GSR

```
CALL PRawBl (Name, Size, Apply, ErrHnd)
```

where

Name is a CHARACTER STRING
Size is a INTEGER*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PRawBloc (  %DESCR Name        : P_VaryingType;
                             Size        : INTEGER;
                      %DESCR AppliedTo   : P_VaryingType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRawBloc (   CONST Name        : STRING;
                             Size        : INTEGER;
                       CONST AppliedTo   : STRING;
                      PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRawBloc( name,size,appliedto )
```

where:

string name,appliedto;
integer size;

DESCRIPTION

This routine creates a structure consisting of a block of contiguous memory with a length of **size** bytes.

PS 390 Command and Syntax

```
Name := RAWBLOCK i;
```

VAX and IBM FORTRAN GSR

```
CALL PRaWRP (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PRaWRP ( PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRaWRP ( PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRaWRP()
```

DESCRIPTION

PRaWRP is used to set the raster mode to write pixel data.

VAX and IBM FORTRAN GSR

```
CALL PRBspl (Name, Order, OpenClosed, NonPeriodic_Periodic, Dimension,
             Nvertices, Vertices, KnotCount, Knots, Chords, ErrHnd)
```

where:

Name is a CHARACTER STRING
Order is an INTEGER*4,
OpenClosed is a LOGICAL*1
NonPeriodic_Periodic is a LOGICAL*1
Dimension is an INTEGER*4 (2D or 3D)
Nvertices is an INTEGER*4
Vertices is defined: REAL*4 Vertices (4,NVert)
Knotcount is a INTEGER*4
Knots is an array (KnotCount + 1) of REAL*4
Chords is an INTEGER*4
ErrHnd user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PRBspl (   %DESCR Name                 : P_VaryingType;
                            Order                : INTEGER;
                            OpenClosed           : BOOLEAN;
                            NonPeriodic_Periodic : BOOLEAN;
                            Dimension            : INTEGER;
                            N_Vertices           : INTEGER;
                        VAR Vertices             : P_VectorListType;
                            KnotCount            : INTEGER;
                        VAR Knots                : P_KnotArrayType;
                            Chords               : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRBspl (     CONST Name                 : STRING;
                             Order                : INTEGER;
                             OpenClosed           : BOOLEAN;
                             NonPeriodic_Periodic : BOOLEAN;
                             Dimension            : INTEGER;
                             N_Vertices           : INTEGER;
                       CONST Vertices             : P_VectorListType;
                             KnotCount            : INTEGER;
                       CONST Knots                : P_KnotArrayType;
                             Chords               : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRBspl(name,order,openclosed,nonperiodic_periodic,dimension,nvertices,
       vertices,knotcount,knots,chords)
```

where:

```
string name;
integer order,dimension,nvertices,knotcount,chords;
boolean openclosed,nonperiodic_periodic;
P_VectorListType vertices;
P_KnotArrayType knots;
```

DESCRIPTION

This routine allows the parametric description of a rational B-spline curve form without having to specify or transfer the coordinates of each constituent vector. It contains the following parametric definitions:

Name specifies the name to be given to the computed rational B-spline.
Order is the order of the curve.
OpenClosed is TRUE for Open and FALSE for Closed.
NonPeriodic_Periodic is TRUE for Nonperiodic and FALSE for Periodic.
Dimension is 2 or 3 (2 or 3 dimensional respectively).
Nvertices specifies the number of vertices.
Vertices specifies the vertices.
KnotCount is the number of knots.
Knots is the knot sequence.
Chords is the number of vectors to be created.

NOTE

None of the parameters in the routine PRBSPL are optional. The dimension must be specified in the PRBSPL routine. In the PS 390 command, dimension is implied by syntax. If KnotCount = 0, then the default knot sequence is generated and the knot array is ignored.

PS 390 Command and Syntax

```
Name  := RATIONAL BSPLINE ORDER = k
         [OPEN/CLOSED]
         [NONPERIODIC/PERIODIC]
         [N = n]
         [VERTICES =]  X1, Y1,[ Z1,] W1
                       X2, Y2,[ Z2,] W2
                       .   .    .    .
                       .   .    .    .
                       XN, YN,[ ZN,] WN
         [KNOTS = t1,t2,...tj]
         CHORDS = q;
```

SEE ALSO

BSPLINE

VAX and IBM FORTRAN GSR

```
CALL PRem (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PRem (   %DESCR Name       : P_VaryingType;
                PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRem (    CONST Name       : STRING;
                PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRem(name)
```

where:

string name;

DESCRIPTION

This routine removes **Name** from the display list.

PS 390 Command and Syntax

```
REMove Name;
```

SEE ALSO

DISPLAY

VAX and IBM FORTRAN GSR

```
CALL PRemFo (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PRemFoll (   %DESCR Name       : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRemFoll (    CONST Name       : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRemFoll(name)
```

where:

string name;

DESCRIPTION

This routine removes a previously placed follower of **Name**.

PS 390 Command and Syntax

```
REMove FOLLOWER of name;
```

SEE ALSO

FOLLOW WITH

VAX and IBM FORTRAN GSR

```
CALL PRemFr (Name1, Name2, ErrHnd)
```

where:

Name1 is a CHARACTER STRING
Name2 is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PRemFrom (   %DESCR Name1    : P_VaryingType;
                       %DESCR Name2    : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRemFrom (    CONST Name1    : STRING;
                        CONST Name2    : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRemFrom(name1,name2)
```

where:

string name1,name2;

DESCRIPTION

This routine removes an instance of a named data structure **Name1** from an instance node **Name2.**

PS 390 Command and Syntax

```
REMove Name1 FROM Name2;
```

SEE ALSO

INCLUDE

VAX and IBM FORTRAN GSR

```
CALL PRemPr (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PRemPref (   %DESCR Name       : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRemPref (    CONST Name    : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRemPref(name)
```

where:

string name;

DESCRIPTION

This routine removes a previously placed prefix.

PS 390 Command and Syntax

```
REMove PREfix of name;
```

SEE ALSO

PREFIX WITH

VAX and IBM FORTRAN GSR

```
CALL PRotX  (Name, Angle, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PRotX (   %DESCR Name         : P_VaryingType;
                           Angle        : REAL;
                    %DESCR AppliedTo    : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRotX (    CONST Name         : STRING;
                           Angle        : SHORTREAL;
                     CONST AppliedTo    : STRING;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRotX(name,angle,appliedto)
```

where:

string name,appliedto;
double angle;

DESCRIPTION

This routine creates a 3x3 rotation matrix that rotates an object **Apply** by **Angle** degrees around the X axis relative to world space origin.

PS 390 Command and Syntax

```
Name := ROTate in X Angle [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PRotY  (Name, Angle, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PRotY (   %DESCR Name        : P_VaryingType;
                           Angle       : REAL;
                    %DESCR AppliedTo   : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRotY (    CONST Name        : STRING;
                           Angle       : SHORTREAL;
                     CONST AppliedTo   : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRotY(name,angle,appliedto)
```

where:

string name,appliedto;
double angle;

DESCRIPTION

This routine creates a 3x3 rotation matrix that rotates an object **Apply** by **Angle** degrees around the Y axis relative to world space origin.

PS 390 Command and Syntax

```
Name := ROTate in Y Angle [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PRotZ  (Name, Angle, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Angle is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PRotZ (   %DESCR Name        : P_VaryingType;
                           Angle       : REAL;
                    %DESCR AppliedTo   : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRotZ (    CONST Name        : STRING;
                           Angle       : SHORTREAL;
                     CONST AppliedTo   : STRING;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRotZ(name,angle,appliedto)
```

where:

string name,appliedto;
double angle;

DESCRIPTION

This routine creates a 3x3 rotation matrix that rotates an object **Apply** by **Angle** degrees around the Z axis relative to world space origin.

PS 390 Command and Syntax

```
Name := ROTate in Z Angle [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PRPoly (Name, Order, Dimension, Coeffs, Chords, ErrHnd)
```

where:

Name is a CHARACTER STRING
Order is an INTEGER*4
Dimension is an INTEGER*4
Coeffs is defined: REAL*4 (4, Order+1)
Chords is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PRPoly (   %DESCR Name       : P_VaryingType;
                            Order      : INTEGER;
                            Dimension  : INTEGER;
                        VAR Coeffs     : P_VectorListType;
                            Chords     : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRPoly (    CONST Name       : STRING;
                            Order      : INTEGER;
                            Dimension  : INTEGER;
                      CONST Coeffs     : P_VectorListType;
                            Chords     : INTEGER;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRPoly(name,order,dimension,coeffs,chords)
```

where:

string name;
integer order,dimension,chords;
P_VectorListType coeffs;

DESCRIPTION

This routine allows the parametric description of many curve forms without having to specify or transfer the coordinates of each constituent vector. It includes the following parametric definitions:

Order is the order of the polynomial.
Dimension is 2 or 3 (2 or 3 dimensions respectively).
Coeffs represent the x,y,z components of the curve.

```
Coeffs [i].V4 [1]:= x(order -i+1)
Coeffs [i].V4 [2]:= y(order -i+1)
Coeffs [i].V4 [3]:= z(order -i+1)
Coeffs [i].V4 [4]:= w(order -i+1
```

To further clarify the description:

Coeffs [1].V4 [1] := the coefficient that will be applied to the t^{order} term.
Coeffs [2].V4 [1] := the coefficient that will be applied to the $t^{order-1}$ term in the resultant x(t) function computed by this command.
:
:
etc.

Chords is the number of vectors to be drawn.

NOTE

The definition for Coeffs in IBM Pascal takes the form *Coeffs(.i.).V4(.1.):=x(order–i+1)*. VAX FORTRAN and IBM FORTRAN take the form *Coeffs(1,i) = x(order–i+1)*. The description of the parameter is otherwise identical for each of the programming languages.

PS 390 Command and Syntax

```
Name  := RATIONAL POLYNOMIAL ORDER = i
         [COEFFICIENTS =] Xi,   Yi,   Zi,   Wi
                          Xi-1, Yi-1, Zi-1, Wi-1
                          :     :     :     :
                          XO,   YO,   ZO,   WO
         CHORDS = q;
```

SEE ALSO

POLYNOMIAL

VAX and IBM FORTRAN GSR

```
CALL PRsvSt (Bytes, ErrHnd)
```

where:.

Bytes is an INTEGER*4
ErrHnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PRsvStor (          Bytes : INTEGER:
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PRsvStor (          Bytes : INTEGER:
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PRsvStor(nbytes)
```

where:

integer nbytes;

DESCRIPTION

This routine is used to reserve working storage space for rendering solids and surfaces. Working storage space must be reserved explicitly using this routine. The parameter **Bytes** represents the number of bytes to be reserved for working storage.

PS 390 Command and Syntax

```
Reserve_Working_Storage size;
```

UNIX/C UTILITY ROUTINE

```
#include <ps300/gsrext.h>

PSavBeg(filename)

where

    char *filename;
```

DESCRIPTION

PSavBeg diverts the output of the GSR library from its current output device to the disk file, **filename**. The Start_of_text character, the byte count, and the mux byte are stripped before saving the data. This is the only and essential difference between using **PSavBeg** and specifying a disk file name as the argument to **PAttach** routine. The output diversion is terminated by a **PSavEnd** call. The data saved thus can be reloaded using the **PLoad** routine.

Since the mux byte is stripped before saving the data on disk, the programmer must ensure that all output saved together use the same mux byte. When the mux byte is expected to change (as when calling **PPutPars** or **PPutG**) **PSavEnd** should be called, and further output should be diverted to a different file by using **PSavBeg** again.

SEE ALSO

PSavEnd, PLoad
Examples using PSavBeg, PSavEnd, and PLoad can be found in *TT3* or in the on-line documentation in **gsrref(3G)**.

UNIX/C UTILITY ROUTINE

```
#include <ps300/gsrext.h>

PSavEnd()
```

DESCRIPTION

PSavEnd terminates the output of the GSR library to a disk file, which was begun by a call to **PSavBeg**. Any further output from the library is sent to the output channel which was in effect before the corresponding **PSavBeg** call. The data saved by the **PSavBeg**, **PSavEnd** combination can be reloaded efficiently, using the **PLoad** routine.

SEE ALSO

PSavBeg, PLoad
Examples using PSavBeg, PSavEnd, and PLoad can be found in *TT3* or in the on-line documentation in **gsrref(3G)**.

VAX and IBM FORTRAN GSR

```
CALL PScale (Name, Vector, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Vector is defined: REAL*4 V(3)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PScaleBy (   %DESCR Name      : P_VaryingType;
                          VAR Vector    : P_VectorType;
                       %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PScaleBy (    CONST Name      : STRING;
                        CONST Vector    : P_VectorType;
                        CONST Name      : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PScaleBy(name,vector,appliedto)
```

where:

string name,appliedto;
P_VectorType vector;

DESCRIPTION

This routine applies a uniform scale transformation matrix to a specified vector list and/or characters specified by **Apply/Applied to**. **Vector** contains the x,y,z scale components.

PS 390 Command and Syntax

```
Name := SCALE by s [APPLied to name1];
Name := SCALE by sx, sy, sz [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PSeBit (Name, BitNumber, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
BitNumber is an INTEGER*4
OnOff is a LOGICAL*1
Apply is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetBit (    %DESCR Name       : P_VaryingType;
                              BitNumber  : INTEGER;
                              OnOff      : BOOLEAN;
                       %DESCR AppliedTo  : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetBit (     CONST Name       : STRING;
                              BitNumber  : INTEGER;
                              OnOff      : BOOLEAN;
                        CONST AppliedTo  : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetBit(name,bitnumber,onoff,appliedto)
```

where:

string name,appliedto;
integer bitnumber;
boolean onoff;

DESCRIPTION

This routine alters one of the 15 global conditional bits during the traversal of the data structure. These conditional bits are initially set to OFF. When the traversal is finished, the bits are restored to their previous values. It contains the following parametric definitions:

BitNumber is an integer from 0 to 14 corresponding to the conditional bit to be set to ON or OFF.

OnOff is TRUE for ON and FALSE for OFF.

PS 390 Command and Syntax

```
Name := SET conditional_BIT n switch [APPLied to name1];
```

SEE ALSO

IF CONDITIONAL_BIT

VAX and IBM FORTRAN GSR

```
CALL PSeChF (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetChrF (    %DESCR Name      : P_VaryingType;
                        %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetChrF (     CONST Name      : STRING;
                         CONST AppliedTo : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetChrF(name,appliedto)
```

where:

string name,appliedto;

DESCRIPTION

This routine sets the type of screen orientation for displayed character strings. When it is used, characters are not affected by rotation or scaling transformations and they are displayed with full size and intensity.

PS 390 Command and Syntax

```
Name := SET CHARacters orientation [APPLied to name1];
```

SEE ALSO

CHARACTERS,
CHARACTER SCALE,
SET CHARACTERS SCREEN_ORIENTED,
SET CHARACTERS WORLD_ORIENTED

VAX and IBM FORTRAN GSR

```
CALL PSeChS (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetChrS (    %DESCR Name      : P_VaryingType;
                        %DESCR AppliedTo : P_VaryingType;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetChrS (  CONST Name      : STRING;
                      CONST AppliedTo : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetChrS(name,appliedto)
```

where:

string name,appliedto;

DESCRIPTION

This routine sets the type of screen orientation for displayed character strings. When it is used, characters are not affected by rotation or scaling trans- formations, but intensity and size will still vary with depth (Z–position).

PS 390 Command and Syntax

```
Name := SET CHARacters orientation [APPLied to name1];
```

SEE ALSO

CHARACTERS,
CHARACTER SCALE,
SET CHARACTERS SCREEN_ORIENTED/FIXED,
SET CHARACTERS WORLD_ORIENTED

VAX and IBM FORTRAN GSR

```
CALL PSeChW (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetChrW (    %DESCR Name      : P_VaryingType;
                        %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetChrW (   CONST Name      : STRING;
                       CONST AppliedTo : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>
PSetChrW(name,appliedto)
```

where:

string name,appliedto;

DESCRIPTION

This routine sets the type of screen orientation for displayed character strings. When it is used, characters are transformed along with any part of the object containing them.

PS 390 Command and Syntax

```
Name := SET CHARacters orientation [APPLied to name1];
```

SEE ALSO

CHARACTERS,
CHARACTER SCALE,
SET CHARACTERS SCREEN_ORIENTED/FIXED

VAX and IBM FORTRAN GSR

```
CALL PSeCns (Cness, Input, Name, ErrHnd)
```

where:

Cness is a LOGICAL
Input is an INTEGER*4
Name is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PSetCnes (          Cness  : BOOLEAN;
                              Input  : INTEGER;
                        %DESCR Name  : P_VaryingType;
                      PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetCnes (          Cness  : BOOLEAN;
                              Input  : INTEGER;
                         CONST Name  : STRING;
                      PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetCnes(cness,input,name)
```

where:

boolean cness;
integer input;
string name;

DESCRIPTION

This routine is used to define a particular function instance input to be a constant or trigger input.

PS 390 Command and Syntax

```
SETUP CNESS TRUE    <i>name;
SETUP CNESS FALSE   <i>name;
```

VAX and IBM FORTRAN GSR

```
CALL PSeCol (Name, Hue, Saturation, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Hue is a REAL*4
Saturation is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PSetColr (   %DESCR Name       : P_VaryingType;
                              Hue        : REAL;
                              Saturation : REAL;
                       %DESCR AppliedTo  : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetColr (    CONST Name       : STRING;
                              Hue        : SHORTREAL;
                              Saturation : SHORTREAL;
                        CONST AppliedTo  : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetColr(name,hue,saturation,appliedto)
```

where:

string name,appliedto;
double hue,saturation;

DESCRIPTION

This routine specifies the color of an object **Apply/AppliedTo**. It contains the following parametric definition:

- Hue is greater than or equal to 0.0 and less than 360.0 with:
  0.0 = pure blue
  120.0 = pure red
  240.0 = pure green
- Saturation is from 0.0 to 1.0 with:
  0.0 = no saturation (white)
  1.0 = full saturation

PS 390 Command and Syntax

```
Name := SET COLOR hue,saturation [APPLied to name1];
```

SEE ALSO

POLYGON

VAX and IBM FORTRAN GSR

```
CALL PSeCon (Name, Contrast, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Contrast is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetCont (   %DESCR Name      : P_VaryingType;
                              Contrast  : REAL;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetCont (    CONST Name      : STRING;
                              Contrast  : SHORTREAL;
                        CONST AppliedTo : STRING;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>
```

```
PSetCont(name,contrast,appliedto)
```

where:

string name,appliedto;
double contrast;

DESCRIPTION

This routine changes the contrast of the data structure **Apply/Applied to**. It contains the following parametric definition:

- Contrast is from 0.0 to 1.0 with:
  0.0 = lowest contrast
  1.0 = highest contrast

PS 390 Command and Syntax

```
Name := SET CONTrast to c [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PSecPl (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING*(*)
Apply is a CHARACTER STRING*(*)
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSecPlan  (   %DESCR Name      : P_VaryingType;
                        %DESCR AppliedTo : P_VaryingType;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSecPlan  (   CONST Name      : STRING;
                        CONST AppliedTo : STRING
                      PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSecPlan(name,appliedto)
```

where:

string name,appliedto;

DESCRIPTION

This routine creates a sectioning-plane operation node specified by **Name**. **Apply/AppliedTo** supplies the name of the entity that contains the polygon defining the sectioning plane.

PS 390 Command and Syntax

```
Name := SECTioning_plane [APPLied to name1];
```

SEE ALSO

POLYGON

VAX and IBM FORTRAN GSR

```
CALL PSeDAl (Name, OnOff, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetDAll  (   %DESCR Name       : P_VaryingType;
                               OnOff      : BOOLEAN;
                        %DESCR AppliedTo  : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetDAll  (   CONST Name       : STRING;
                              OnOff      : BOOLEAN;
                        CONST AppliedTo  : STRING
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetDAll(name,onoff,appliedto)
```

where:

string name,appliedto;
boolean onoff;

DESCRIPTION

This routine sets all display(s) ON or OFF. **Onoff** is TRUE for ON and FALSE for OFF.

PS 390 Command and Syntax

```
Name := SET DISPlays ALL switch [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PSeDCL (Name, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1 defined: .TRUE. for On and .FALSE. for Off.
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetDCL (   %DESCR Name       : P_VaryingType;
                             OnOff      : BOOLEAN;
                      %DESCR Name       : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetDCL (    CONST Name       : STRING;
                             OnOff      : BOOLEAN;
                       CONST AppliedTo  : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetDCL(name,onoff,appliedto)
```

where:

string name,appliedto;
boolean onoff;

DESCRIPTION

With depth clipping on (TRUE), data between the eye and the front clipping plane will be clipped, data between the front clipping plane and back clipping plane will appear with an intensity gradient, and data behind the back clipping plane will be clipped.

With depth clipping off (FALSE), data between the eye and front clipping plane will appear at full intensity, data between the front clipping plane and back clipping plane will appear with an intensity gradient, and data behind the back clipping plane will appear at minimum intensity.

PS 390 Command and Syntax

```
Name := SET DEPTH_CLipping switch [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PSeDOF (Name, OnOff, N, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1
N is an INTEGER*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetDOnF  (   %DESCR Name       : P_VaryingType;
                              OnOff       : BOOLEAN;
                               N          : INTEGER;
                          %DESCR AppliedTo : P_VaryingType;
                        PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetDOnF  (   CONST Name       : STRING;
                             OnOff       : BOOLEAN;
                             N           : INTEGER;
                          CONST AppliedTo : STRING
                        PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetDOnF(name,onoff,n,appliedto)
```

where:

string name,appliedto;
boolean onoff;
int n;

DESCRIPTION

This routine specifies the display number n to be set to ON or Off. **Onoff** is TRUE for ON and FALSE for OFF.

PS 390 Command and Syntax

```
Name := SET DISPlay n[,m...] switch [APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PSeInt (Name, OnOff, IMin, IMax, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1
IMin is a REAL*4
IMax is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetInt (    %DESCR Name       : P_VaryingType;
                              OnOff      : BOOLEAN;
                              Imin       : REAL;
                              Imax       : REAL;
                       %DESCR AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetInt (     CONST Name       : STRING;
                              OnOff      : BOOLEAN;
                              Imin       : SHORTREAL;
                              Imax       : SHORTREAL;
                        CONST AppliedTo  : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetInt(name,onoff,imin,imax,appliedto)
```

where:

```
string name,appliedto;
boolean onoff;
double imin,imax;
```

DESCRIPTION

This routine specifies the intensity variation for depth cueing and has the following parametric definition:

- **OnOff** TRUE for On and FALSE for Off.
- **IMin** is a real number from 0.0 to 1.0 that represents the dimmest intensity setting
- **IMax** is a real number from 0.0 to 1.0 that represents the brightest intensity setting.

PS 390 Command and Syntax

```
Name := SET INTENsity switch imin:imax [APPLied to name1];
```

SEE ALSO

VIEWPORT

VAX and IBM FORTRAN GSR

```
CALL PSeLnt (Name, Pattern, Continuous, Apply, Errhnd)
```

where:

Name is a CHARACTER STRING
Pattern is an INTEGER*4
Continuous is a LOGICAL*1
Apply is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PSetLinT  (   %DESCR  Name          : P_VaryingType
                                Pattern       : INTEGER
                                Continuous    : BOOLEAN
                        %DESCR  AppliedTo     : P_VaryingType
                      PROCEDURE Error_Handler (Err : INTEGER)
```

IBM PASCAL GSR

```
PROCEDURE PSetLinT  (   CONST  Name          : STRING
                                Pattern       : INTEGER
                                Continuous    : BOOLEAN
                        CONST  AppliedTo     : STRING
                      PROCEDURE Error_Handler (Err : INTEGER)
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetLinT(name,pattern,continuous,appliedto)
```

where:

string name,appliedto;
integer pattern;
boolean continuous;

DESCRIPTION

This routine specifies the line texture pattern to be used in drawing the vector lists that appear below the node created by this command. There are up to 127 hardware-generated line textures possible. The parameter **Pattern** is an integer between 1 and 127. The desired line texture is indicated by the setting or clearing of the lower 7-bit positions in **Pattern** when represented in binary. An individual pattern unit is 1.1 centimeters in length. Some of the more common patterns and their corresponding bit settings are shown below:

```
Pattern  Bit representation  Line Texture repeated twice
 127         1111111          --------------     Solid
 124         1111100          -----  -----       Long Dashed
 122         1111010          ---- - ---- -      Long Short Dashed
 106         1101010          -- - - -- - -      Long Short Short Dashed
```

Continuous is a LOGICAL value used to set a flag to indicate if the specified line texture should continue from one vector to the next. If TRUE, the line texture will continue from one vector to the next through the endpoint. If FALSE, the line texture will start and stop at the vector endpoints.

Pattern is an integer between 1 and 127 that specifies the desired line texture. When **Pattern** is less that 1 or greater than 127, solid lines are produced.

Apply/AppliedTo is the name of the structure to which the line texture is applied.

The default line texture is a solid line.

NOTES

Since 7 bit positions are used, it is not possible to create a symmetric pattern. When line-texturing is applied to a vector, the vector that is specified is displayed as a patterned, rather that solid line. If the line is smaller than the pattern length, then as much of the pattern that can be displayed with the vector is displayed. If the line is smaller than the smallest element of the pattern, then the line is displayed as solid.

The With Pattern and curve commands create multiple vectors in memory. To the line-texturing hardware, each vector in a pattern or curve is seen as an individual vector. Line-texturing a patterned line or curve is the same as line-texturing a number of small segments. Curves and patterns affect line-texturing only in that they tend to create short vectors that may be too short to be completely textured.

PS 390 Command and Syntax

```
name := SET LINe_texture [AROUnd_corners] pattern [APPLied to name1];
```

SEE ALSO

PATTERN, PATTERN WITH

VAX and IBM FORTRAN GSR

```
CALL PSeLOD (Name, Level, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Level is an INTEGER*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetLOD (   %DESCR Name      : P_VaryingType;
                             Level     : INTEGER;
                      %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetLOD (    CONST Name      : STRING;
                             Level     : INTEGER;
                       CONST AppliedTo : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetLOD(name,level,appliedto)
```

where:

string name,appliedto;
integer level;

DESCRIPTION

This routine alters a global level-of-detail value temporarily. These temporary settings allow for conditional referencing to other data structures. When the traversal of data is finished, the level of detail is restored to its original level. **Level** is an integer from 0 to 32767 that indicates the level-of-detail value.

PS 390 Command and Syntax

```
Name := SET LEVel_of_detail TO n [APPLied to name1];
```

SEE ALSO

DECREMEMT LEVEL_OF_DETAIL, IF LEVEL_OF_DETAIL, INCREMENT LEVEL_OF_DETAIL

VAX and IBM FORTRAN GSR

```
CALL PSePID (Name, PickId, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
PickId is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetPID (    %DESCR Name      : P_VaryingType;
                       %DESCR PickId    : P_VaryingType;
                       %DESCR AppliedTo : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetPID (     CONST Name      : STRING;
                        CONST PickId    : STRING;
                        CONST AppliedTo : STRING;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>
```

```
PSetPID(name,pickid,appliedto)
```

where:

string name,pickid,appliedto;

DESCRIPTION

This routine specifies textual information (e.g. a character string) **pickid** that will be reported back if a pick occurs anywhere on the specified display structure **Apply/AppliedTo.**

PS 390 Command and Syntax

```
Name := SET PICKing IDentifier = id_name [APPLied to name1];
```

SEE ALSO

SET PICKING, SET PICKING LOCATION

VAX and IBM FORTRAN GSR

```
CALL PSePLo (Name, Xcenter, Ycenter, Xsize, Ysize, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
XCenter, YCenter are REAL*4
Xsize, Ysize are REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetPLoc (   %DESCR Name      : P_VaryingType;
                              Xcenter   : REAL;
                              Ycenter   : REAL;
                              Xsize     : REAL;
                              Ysize     : REAL;
                         %DESCR AppliedTo : P_VaryingType;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetPLoc (    CONST Name      : STRING;
                              Xcenter   : SHORTREAL;
                              Ycenter   : SHORTREAL;
                              Xsize     : SHORTREAL;
                              Ysize     : SHORTREAL;
                          CONST AppliedTo : STRING;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetPLoc(name,xcenter,ycenter,xsize,ysize,appliedto)
```

where:

string name,appliedto;
double xcenter,ycenter,xsize,ysize;

DESCRIPTION

This routine specifies a rectangular picking area at **x,y** within the current viewport. It contains the following parametric definitions:

Xcenter, Ycenter signify the center of the pick location.
Xsize, Ysize specify the boundaries of the pick rectangle.

PS 390 Command and Syntax

```
Name  := SET PICKING LOCation = X,Y size_x, size_y;
```

SEE ALSO

SET PICKING, SET PICKING IDENTIFIER

VAX and IBM FORTRAN GSR

```
CALL PSePOf (Name, OnOff, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
OnOff is a LOGICAL*1
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetPOnf (   %DESCR Name       : P_VaryingType;
                              OnOff      : BOOLEAN;
                       %DESCR AppliedTo  : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetPOnf (    CONST Name       : STRING;
                              OnOff      : BOOLEAN;
                        CONST AppliedTo  : STRING;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetPOnf(name,onoff,appliedto)
```

where:

string name,appliedto;
boolean onoff;

DESCRIPTION

This routine enables/disables picking for a specified display structure **Apply/AppliedTo**. **OnOff** is TRUE for picking enabled and FALSE for disabled.

PS 390 Command and Syntax

```
Name := SET PICKing switch [APPLied to name1];
```

SEE ALSO

SET PICKING IDENTIFIER, SET PICKING LOCATION

VAX and IBM FORTRAN GSR

```
CALL PSeR (Name, PhaseOn, PhaseOff, InitOnOff, Delay, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
PhaseOn is an INTEGER*4
PhaseOff is an INTEGER*4
InitOnOff is a LOGICAL*1 defined: .TRUE. for On and .FALSE. for Off
Delay is an INTEGER*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetR (    %DESCR Name       : P_VaryingType;
                            PhaseOn    : INTEGER;
                            PhaseOff   : INTEGER;
                            InitOnOff  : BOOLEAN;
                            Delay      : INTEGER;
                       %DESCR AppliedTo : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetR (     CONST Name       : STRING;
                            PhaseOn    : INTEGER;
                            PhaseOff   : INTEGER;
                            InitOnOff  : BOOLEAN;
                            Delay      : INTEGER;
                        CONST AppliedTo : STRING;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetR(name,phaseon,phaseoff,initonoff,delay,appliedto)
```

where:

string name,appliedto;
integer phaseon,phaseoff,delay;
boolean initonoff;

DESCRIPTION

This routine is used to control blinking of display structures. It temporarily sets the duration of the two global ON and OFF phases of the PHASE attribute. Each duration is specified in number of refresh frames. The default phase is OFF and never changes unless a SET RATE node is encountered. The routine has the following parametric definitions:

PhaseOn designates the duration of the ON phase.
PhaesOff designates the duration of the OFF phase.
InitOnOff specifies the initial state of Phase:TRUE for ON, FALSE for OFF.
Delay is the number of refresh frames in the initial state.

The Phase attribute is usually tested further down the display tree using IF PHASE to conditionally display a data structure.

PS 390 Command and Syntax

```
Name := SET RATE phase_on phase_off [initial_state] [delay]
        [APPLied to name1];
```

SEE ALSO

IF PHASE, SET RATE EXTERNAL

VAX and IBM FORTRAN GSR

```
CALL PSeREx (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSetRExt (    %DESCR Name      : P_VaryingType;
                        %DESCR AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSetRExt (     CONST Name      : STRING;
                         CONST AppliedTo : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSetRExt(name,appliedto)
```

where:

string name,appliedto;

DESCRIPTION

This routine sets up a data structure that can be used to alter the PHASE attribute using an external source, such as a function network or a message from the host computer. This is in contrast to the SET RATE routine where the PHASE attribute is changed based on refresh cycles.

PS 390 Command and Syntax

```
Name := SET RATE EXTernal [APPLied to name1];
```

SEE ALSO

IF PHASE, SET RATE

VAX and IBM FORTRAN GSR

```
CALL PSnBoo (B, Input, Destination, ErrHnd)
```

where:

B is .TRUE. or .FALSE., the logical value to be sent

*Input is an INTEGER*4 corresponding to the input of the display structure, function instance, or variable, Destination

Destination is a CHARACTER STRING representing the destination

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndBool (           B           : BOOLEAN;
                               Input       : INTEGER;
                    %DESCR Destination : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndBool (          B           : BOOLEAN;
                              Input       : INTEGER;
                        CONST Destination : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndBool(b,input,destination)
```

where:

boolean b;
integer input;
string destination;

DESCRIPTION

This routine sends a Boolean value to **Input** of a specified function instance, display structure, or variable **Destination.**

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

```
                FORTRAN

Mnemonic      <Input>      INTEGER*4_Value

PILAST        <LAST>             -5


           Pascal & UNIX

Mnemonic      <Input>      INTEGER*4_Value

P_Last        <LAST>             -5
```

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnFix (i, Input, Destination, ErrHnd)
```

where:

i is an INTEGER*4, the integer to be sent

*Input is an INTEGER*4 corresponding to the input of a display structure, function instance, or variable destination

Destination is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndFix (          i           : INTEGER;
                             Input       : INTEGER;
                        %DESCR Destination : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndFix (          i           : INTEGER;
                             Input       : INTEGER;
                         CONST Destination : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndFix(i,input,destination)
```

where:

integer i;
integer input;
string destination;

DESCRIPTION

This routine sends the value of **i** to the specified **Input** of the display structure or function instance **Destination.**

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

	FORTRAN	
Mnemonic	<Input>	INTEGER*4_Value
PIDEL	<DELETE>	-1
PICLR	<CLEAR>	-2
	Pascal & UNIX	
P_Delete	<DELETE>	-1
P_Clear	<CLEAR>	-2

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnM2d (Matrix, Input, Destination, ErrHnd)
```

where:

Matrix is the matrix to be sent and is defined: REAL*4 Mat(4,4)

Input is an INTEGER*4 corresponding to the input of a variable, function instance or display structure

Destination is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndM2d (      VAR Matrix       : P_MatrixType;
                             Input        : INTEGER;
                        %DESCR Destination : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndM2d (    CONST Matrix       : P_MatrixType;
                             Input        : INTEGER;
                       CONST Destination  : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndM2d(matrix,input,destination)
```

where:

P_MatrixType matrix;
integer input;
string destination;

DESCRIPTION

This routine sends a 2x2 **Matrix** to the specified **Input** of a display structure, function instance, or variable **Destination.**

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnM3d (Matrix, Input, Destination, ErrHnd)
```

where:

Matrix is the matrix to be sent and is defined: REAL*4 Mat(4,4)

Input is an INTEGER*4 corresponding to the input of a variable, function instance or display structure

Destination is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndM3d (      VAR Matrix      : P_MatrixType;
                             Input       : INTEGER;
                        %DESCR Destination: P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndM3d (    CONST Matrix      : P_MatrixType;
                             Input       : INTEGER;
                       CONST Destination : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndM3d(matrix,input,destination)
```

where:

P_MatrixType matrix;
integer input;
string destination;

DESCRIPTION

This routine sends a 3x3 **Matrix** to the specified **Input** of a display structure, function instance, or variable **Destination.**

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnM4d (Matrix, Input, Destination, ErrHnd)
```

where:

Matrix is the matrix to be sent and is defined: REAL*4 Mat(4,4)

Input is an INTEGER*4 corresponding to the input of a variable, function instance or display structure

Destination is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndM4d (      VAR Matrix      : P_MatrixType;
                             Input       : INTEGER;
                        %DESCR Destination: P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndM4d (    CONST Matrix       : P_MatrixType;
                             Input        : INTEGER;
                       CONST Destination  : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndM4d(matrix,input,destination)
```

where:

P_MatrixType matrix;
integer input;
string destination;

DESCRIPTION

This routine sends a 4x4 **Matrix** to the specified **Input** of a display structure, function instance, or variable **Destination.**

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnPL  (Count, DrawMove, Input, Destination, ErrHnd)
```

where:

Count is an INTEGER*4
DrawMove is LOGICAL*1 and is defined: .TRUE. is Draw and .FALSE. is Move
Input is an INTEGER*4
Destination is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndPL (          Count       : INTEGER;
                            DrawMove    : BOOLEAN;
                            Input       : INTEGER;
                   %DESCR   Name        : P_VaryingType;
                 PROCEDURE  Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndPL (          Count       : INTEGER;
                            DrawMove    : BOOLEAN;
                            Input       : INTEGER;
                    CONST   Name        : STRING;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndPL(count,drawmove,input,name)
```

where:

integer count,input;
boolean drawmove;
string name;

DESCRIPTION

This routine sends **count** number of **Draw** or **Move** specifications to consecutive vectors beginning at vector **Input** of the vector list **Destination. DrawMove** is TRUE for Draw and FALSE for Move.

PS 390 Command and Syntax

```
SEND number*mode TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnRea (Real, Input, Destination, ErrHnd)
```

where:

Real is the REAL*4 to be sent
Input is an INTEGER*4
Destination is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndReal (          real          : REAL;
                              Input         : INTEGER;
                     %DESCR Destination     : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndReal (          real          : SHORTREAL;
                              Input         : INTEGER;
                       CONST Destination    : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndReal(real,input,destination)
```

where:

double real;
integer input;
string destination;

DESCRIPTION

This routine sends a real number **Real** to a specified **Input** of a display structure or function instance **Destination.**

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnRSt (String, Input, Destination, ErrHnd)
```

where:

String is a CHARACTER STRING
*Input is an INTEGER*4
Destination is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndRStr (   %DESCR String      : P_VaryingType;
                             Input        : INTEGER;
                       %DESCR Destination : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndRStr (    CONST String      : STRING;
                             Input        : INTEGER;
                        CONST Destination : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

There is no corresponding UNIX routine.

DESCRIPTION

This routine does NOT translate the character **String**. If the character string = CHR(0) // CHR(1), then CHR(0) // CHR(1) is sent as the string. This routine is functionally identical to the SEND STRING call for VAX systems. For IBM systems no translation from EBCDIC to ASCII is performed on the string. This routine should be used when a character string of some length containing arbitrary characters is to be sent to a function network without translation. An example of where SEND RAW STRING must be used is as follows.

Where the PS 390 command to send a string would be,

```
SEND CHAR (1) to <2> CONSTANT1;
```

the equivalent Graphics Support Routine call would be,

```
String = Char (1)
CALL PSnRSt (Str, 2, 'CONSTANT1', ErrHnd)
```

where **String** is declared CHARACTER STRING*1

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

FORTRAN

Mnemonic	<Input>	INTEGER*4_Value
PILAST	<LAST>	-5
PISUBS	<SUBSTITUTE>	-6

Pascal & UNIX

Mnemonic	<Input>	INTEGER*4_Value
P_Last	<LAST>	-5
P_Substitute	<SUBSTITUTE>	-6

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnSt  (String, Input, Destination, ErrHnd)
```

where:

String is a CHARACTER STRING to be sent
*Input is an INTEGER*4
Destination is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndStr (    %DESCR String      : P_VaryingType;
                             Input        : INTEGER;
                       %DESCR Destination : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndStr (    CONST String      : STRING;
                             Input       : INTEGER;
                       CONST Destination : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndStr(string,input,destination)
```

where:

string string;
integer input;
string destination;

DESCRIPTION

This routine sends the character **String** to the **Input** of a display structure **Destination.** (On VAX systems no translation is required. On IBM systems the **String** is translated from EBCDIC to ASCII.)

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

FORTRAN

Mnemonic	<Input>	INTEGER*4_Value
PILAST	<LAST>	-5
PISUBS	<SUBSTITUTE>	-6

Pascal & UNIX

Mnemonic	<Input>	INTEGER*4_Value
P_Last	<LAST>	-5
P_Substitute	<SUBSTITUTE>	-6

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnV2d (Vector, Input, Destination, ErrHnd)
```

where:

Vector is the vector to be sent and is defined: REAL*4 V(2)

*Input is an INTEGER*4 corresponding to the input of a function instance, a variable, or a display structure

Destination is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndV2d (       VAR Vector      : P_VectorType;
                              Input       : INTEGER;
                         %DESCR Destination : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndV2d (    CONST Vector      : P_VectorType;
                              Input       : INTEGER;
                         CONST Destination : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndV2d(vector,input,destination)
```

where:

P_VectorType vector;
integer input;
string destination;

DESCRIPTION

This routine sends a 2D vector to the specified **Input** of a display structure, function instance, or variable **Destination.**

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

FORTRAN

Mnemonic	<Input>	INTEGER*4_Value
PIAPP	<APPEND>	0
PISTEP	<STEP>	-3
PIPOS	<POSITION>	-4
PILAST	<LAST>	-5

Pascal & UNIX

Mnemonic	<Input>	INTEGER*4_Value
P_Append	<APPEND>	0
P_Step	<STEP>	-3
P_Position	<POSITION>	-4
P_Last	<LAST>	-5

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnV3d (Vector, Input, Destination, ErrHnd)
```

where:

Vector is the vector to be sent and is defined: REAL*4 V(3)

*Input is an INTEGER*4 corresponding to the input of a function instance, a variable, or a display structure

Destination is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndV3d (       VAR Vector      : P_VectorType;
                              Input       : INTEGER;
                         %DESCR Destination : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndV3d (    CONST Vector      : P_VectorType;
                              Input       : INTEGER;
                         CONST Destination : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndV3d(vector,input,destination)
```

where:

P_VectorType vector;
integer input;
string destination;

DESCRIPTION

This routine sends a 3D vector to the specified **Input** of a display structure, function instance, or variable **Destination.**

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

FORTRAN

Mnemonic	<Input>	INTEGER*4_Value
PIAPP	<APPEND>	0
PISTEP	<STEP>	-3
PIPOS	<POSITION>	-4
PILAST	<LAST>	-5

Pascal & UNIX

Mnemonic	<Input>	INTEGER*4_Value
P_Append	<APPEND>	0
P_Step	<STEP>	-3
P_Position	<POSITION>	-4
P_Last	<LAST>	-5

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnV4d (Vector, Input, Destination, ErrHnd)
```

where:

Vector is the vector to be sent and is defined: REAL*4 V(4)

*Input is an INTEGER*4 corresponding to the input of a function instance, a variable, or a display structure

Destination is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndV4d (       VAR Vector      : P_VectorType;
                              Input       : INTEGER;
                         %DESCR Destination : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndV4d (    CONST Vector       : P_VectorType;
                             Input        : INTEGER;
                       CONST Destination  : STRING;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndV4d(vector,input,destination)
```

where:

P_VectorType vector;
integer input;
string destination;

DESCRIPTION

This routine sends a 4D vector to the specified **Input** of a display structure, function instance, or variable **Destination.**

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

FORTRAN

Mnemonic	<Input>	INTEGER*4_Value
PIAPP	<APPEND>	0
PISTEP	<STEP>	-3
PIPOS	<POSITION>	-4
PILAST	<LAST>	-5

Pascal & UNIX

Mnemonic	<Input>	INTEGER*4_Value
P_Append	<APPEND>	0
P_Step	<STEP>	-3
P_Position	<POSITION>	-4
P_Last	<LAST>	-5

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnVal (VarNam, Input, Destination, ErrHnd)
```

where:

VarNam is a CHARACTER STRING that is the name of the Variable

*Input is an INTEGER*4 corresponding to the input of a function instance, a variable, or a display structure

Destination is a CHARACTER STRING

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndVal (     %DESCR Varnam        : P_VaryingType;
                              Input          : INTEGER;
                        %DESCR Destination   : P_VaryingType;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndVal (      CONST Varnam        : STRING;
                              Input          : INTEGER;
                         CONST Destination   : STRING;
                   PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndVal(varname,input,destination)
```

where:

string varname;
integer input;
string destination;

DESCRIPTION

This routine sends the current value in the variable **VarNam** to a designated **Input** of a display structure or function instance **Destination.**

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

FORTRAN

Mnemonic	<Input>	INTEGER*4_Value
PIAPP	<APPEND>	0
PIDEL	<DELETE>	-1
PICLR	<CLEAR>	-2
PISTEP	<STEP>	-3
PIPOS	<POSITION>	-4
PILAST	<LAST>	-5
PISUBS	<SUBSTITUTE>	-6

Pascal & UNIX

Mnemonic	<Input>	INTEGER*4_Value
P_Append	<APPEND>	0
P_Delete	<DELETE>	-1
P_Clear	<CLEAR>	-2
P_Step	<STEP>	-3
P_Position	<POSITION>	-4
P_Last	<LAST>	-5
P_Substitute	<SUBSTITUTE>	-6

PS 390 Command and Syntax

```
SEND option TO <n>name1;
```

VAX and IBM FORTRAN GSR

```
CALL PSnVL  (Name1, Input, Name2, ErrHnd)
```

where:

Name1 is a CHARACTER STRING containing the name of the vector list to be sent

*Input is an INTEGER*4 corresponding to the index of the first vector to be replaced in (Name2) with the vectors from (Name1)

Name2 is a CHARACTER STRING containing the name of the destination of the vector list

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSndVL (    %DESCR  Name1    : P_VaryingType;
                              Input    : INTEGER;
                      %DESCR  Name2    : P_VaryingType;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSndVL (     CONST  Name1    : STRING;
                              Input    : INTEGER;
                       CONST  Name2    : STRING;
                  PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSndVL(name1,input,name2)
```

where:

```
string name1,name2;
integer input;
```

DESCRIPTION

This routine replaces the vectors beginning at vector **Input** of the vector list **Name2** with the vectors from vector list **Name1** .

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

FORTRAN

Mnemonic	<Input>	INTEGER*4_Value
PIAPP	<APPEND>	0
PILAST	<LAST>	-5

Pascal & UNIX

Mnemonic	<Input>	INTEGER*4_Value
P_Append	<APPEND>	0
P_Last	<LAST>	-5

PS 390 Command and Syntax

```
SEND VL (Name1) TO <i> Name2;
```

VAX and IBM FORTRAN GSR

```
CALL PSolRe (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING*(*)
Apply is a CHARACTER STRING*(*)
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSolRend (    %DESCR  Name      : P_VaryingType;
                        %DESCR  AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSolRend (    CONST Name      : STRING;
                        CONST AppliedTo : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSolRend(name, appliedto)
```

where:

string name, appliedto;

DESCRIPTION

This routine defines a solid-rendering operation node, marking its descendent structure so that solid renderings can be performed on it. **Name** supplies the name to be given to the solid-rendering operation node. **Apply/AppliedTo** supplies the name of the entity that this operation node will be applied to.

PS 390 Command and Syntax

```
Name := SOLID_RENDERING [Applied to name1];
```

SEE ALSO

SURFACE_RENDERING

VAX and IBM FORTRAN GSR

```
CALL PStdFo (Name, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PStdFont (    %DESCR  Name      : P_VaryingType;
                        %DESCR  AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PStdFont (    CONST Name      : STRING;
                        CONST AppliedTo : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PStdFont(name,appliedto)
```

where:

string name,appliedto;

DESCRIPTION

This routine establishes the standard PS 390 character font as the working font.

PS 390 Command and Syntax

Name := STANdard FONT [APPLied to name1];

SEE ALSO

CHARACTER FONT

VAX and IBM FORTRAN GSR

```
CALL PSURGB (Red, Green, Blue, Hue, Saturation, Intensity)
```

where:

Red, Green, Blue are INTEGER*4
Hue, Saturation, Intensity are REAL*4

VAX and IBM Pascal GSR

```
PROCEDURE PSUTIL_HSIRGB (  VAR red, green blue            :  INTEGER;
                           VAR Hue, Saturation, Intensity :  REAL);
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSUTIL_HSIRGB(red,green,blue,hue,saturation,intensity)
```

where:

integer *red,*green,*blue;
double hue,saturation,intensity;

DESCRIPTION

This procedure converts Hue, Saturation, and Intensity color specifications to Red, Green, and Blue color specification. For a given Hue, Saturation, and Intensity the routine returns RGB values as integers between 0 and 255, and uses a color wheel where a Hue of 0 is blue, a Hue of 120 is red, and a Hue of 240 is green.

VAX and IBM FORTRAN GSR

```
CALL PSurRe (Name, Apply, ErrHnd) where:
```

Name is a CHARACTER STRING*(*)
Apply is a CHARACTER STRING*(*)
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PSurRend (    %DESCR  Name      : P_VaryingType;
                        %DESCR  AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PSurRend (    CONST Name      : STRING;
                        CONST AppliedTo : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PSurRend(name, appliedto)
```

where:

string name, appliedto;

DESCRIPTION

This routine defines a surface-rendering operation node, marking its descendent structure so that surface renderings can be performed on it. **Name** supplies the name to be given to the surface-rendering operation node. **Apply/AppliedTo** supplies the name of the entity that this operation node will be applied to.

PS 390 Command and Syntax

```
Name := SURFACE_RENDERING [Applied to name1];
```

SEE ALSO

SOLID_RENDERING

VAX and IBM FORTRAN GSR

```
CALL PTrans (Name, Vector, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
Vector is the vector containing the x,y,z translation values and is defined: REAL*4 V(3)
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PTransBy (     %DESCR  Name       : P_VaryingType;
                             VAR Vector     : P_VectorType
                         %DESCR  AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PTransBy (      CONST Name       : STRING;
                          CONST Vector     : P_VectorType;
                          CONST AppliedTo  : STRING;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PTransBy(name,vector,appliedto)
```

where:

string name,appliedto;
P_VectorType vector[1];

DESCRIPTION

This routine applies a translation vector to the specified data structure **Apply/AppliedTo**.

```
FORTRAN                          UNIX
V(1) = x translation             Vec.V4[0]:= x translation
V(2) = y translation             Vec.V4[1]:= y translation
V(3) = z translation             Vec.V4[2]:= z translation

VAX Pascal                       IBM Pascal
Vec.V4[1]:= x translation        Vec.V4(.1.):= x translation
Vec.V4[2]:= y translation        Vec.V4(.2.):= y translation
Vec.V4[3]:= z translation        Vec.V4(.3.):= z translation
```

NOTE

All 3 Vector components must be specified. Z is not optional in the GSR.

PS 390 Command and Syntax

```
Name := TRANslate by tx,ty[,tz][APPLied to name1];
```

VAX and IBM FORTRAN GSR

```
CALL PVar (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING containing the name of the variable to be created. ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PVar (    %DESCR  Name     : P_VaryingType;
                PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PVar (    CONST Name       : STRING;
                PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PVar(name)
```

where:

string name;

DESCRIPTION

This routine defines a PS 390 variable, where **Name** contains the name of the variable to be created.

PS 390 Command and Syntax

```
VARiable Name1[,name2 ... namen];
```

SEE ALSO

SEND VALUE

VAX and IBM FORTRAN GSR

```
CALL PVcBeg (Name, VectorCount, BlockNormalized, ColorBlend, Dimension,
Class, ErrHnd)
```

where:

Name is a CHARACTER STRING defining the name of the vector list

VectorCount is an INTEGER*4 specifying the total number of vectors in the vector list

BlockNormalized is a LOGICAL*1 defined: .TRUE. for Block Normalized, .FALSE. for Vector Normalized

ColorBlend is a LOGICAL*1 defined: .TRUE. for Color Blending, .FALSE. for normal depth cueing

Dimension is an INTEGER*4 2 or 3 (2 or 3 dimensions respectively)

*Class is an INTEGER*4 defining the class of the vector list

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PVecBegn (  %DESCR  Name           : P_VaryingType;
                              VectorCount    : INTEGER;
                              BlockNormalized: BOOLEAN;
                              ColorBlending  : BOOLEAN;
                              Dimension      : INTEGER;
                              Class          : INTEGER;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PVecBegn (    CONST Name           : STRING;
                              VectorCount    : INTEGER;
                              BlockNormalized: BOOLEAN;
                              ColorBlending  : BOOLEAN;
                              Dimension      : INTEGER;
                              Class          : INTEGER;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PVecBegn(name,vectorcount,blocknormalized,colorblending,dimension,class)
```

where:

```
string name;
integer vectorcount,dimension,class;
boolean blocknormalized,colorblending;
```

DESCRIPTION

This routine must be called to begin a vector list. To send a vector list, the user must call the routines for Begin, List, and End.

NOTE

The dimension must be specified in the application routine. In the PS 390 command, dimension is implied by syntax.

* These mnemonics may be referenced directly by the user if the file containing the declarations is INCLUDED in the routine. See Section *TT3 Using the GSRs* for a description of this file. A description of inputs to display structures and their INTEGER*4 value is given below.

	FORTRAN	
Mnemonic	Meaning	INTEGER*4_Value
PVCONN	Connected	0
PVDOTS	Dots	1
PVITEM	Itemized	2
PVSEPA	Separate	3
PVTAB	Tabulated	4

Pascal & UNIX

Mnemonic	Meaning	INTEGER*4_Value
P_Conn	Connected	0
P_Dots	Dots	1
P_Item	Itemized	2
P_Sepa	Separate	3
P_Tab	Tabulated	4

Together, the Begin, List, and End routines implement the PS 390 command:

PS 390 Command and Syntax

```
Name  := VECTOR_LIST [options][N=n]vectors;
```

VAX and IBM FORTRAN GSR

```
CALL PVcEnd (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PVecEnd  (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PVecEnd  (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PVecEnd()
```

DESCRIPTION

This routine must be called to end a vector list. To send a vector list, the user must call the routines for Begin, List, and End.

PS 390 Command and Syntax

```
Name  := VECTOR_LIST [options][N=n]vectors;
```

VAX and IBM FORTRAN GSR

```
CALL PVcLis (NumberOfVectors,Vectors,PosLin,ErrHnd)
```

where:

NumberOfVectors is the number of vectors in the vector list and is defined: INTEGER*4
Vectors is the array containing the vectors of the vector list and is defined: REAL*4 (4, NVec) where:

Vectors(1,n) = vector n x-component
Vectors(2,n) = vector n y-component
Vectors(3,n) = vector n z-component
Vectors(4,n) = vector n intensity (or hue)
0 <= Vectors(4,n) <=1 or
0 <= Vectors(4,n) <=127 if the vector list is tabulated.

PosLin is the array containing the move/position - draw/line information for each vector.
PosLin is defined : LOGICAL*1 PosLin(NVec)
If PosLin(n) = .TRUE. then vector n is a draw(line) vector.
If PosLin(n) = .FALSE. then vector n is a move(position) vector.

ErrHnd is the user-defined error-handler subroutine.

NOTE - FORTRAN

The POSLIN Array is always required, however the CLASS specified in PVcBeg determines how it is used. For CONNECTED, DOTS, and SEPARATE, the user need not specify the contents of POSLIN. For ITEMIZED, the user-specified position/line is used.

VAX PASCAL GSR

```
PROCEDURE PVecList (          NumberOfVectors : INTEGER;
                         CONST Vectors        : P_VectorListType;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PVecList (          NumberOfVectors : INTEGER;
                          VAR Vectors         : P_VectorListType;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PVecList(numberofvectors,vectors)
```

where:

```
integer numberofvectors;
P_VectorListType vectors;
```

DESCRIPTION

This routine must be called to send a piece of a vector list. For vector-normalized vector lists, this procedure can be called repeatedly to send the vector list down in pieces. Multiple calls to this procedure are not permitted for the block-normalized vector list case, unless the procedure PVecMax (PVcMax) is called first. To send a vector list, the user must call the routines for Begin, List, and End.

Vectors is the array containing the vectors of the vector list. The format of this parameter is shown below for VAX Pascal, IBM Pascal and UNIX.

VAX Pascal

```
Vectors [n].V4[1]:= vector n x-component
Vectors [n].V4[2]:= vector n y-component
Vectors [n].V4[3]:= vector n z-component
Vectors [n].V4[4]:= vector n intensity (or hue)
   0 <= Vectors [n].V4[4] <=1 or
   0 <= Vectors [n].V4[4] <=127 if the vector list is tabulated.
Vectors [n].draw :=TRUE if vector n is a draw/line vector.
Vectors [n].draw :=FALSE if vector n is a move/position vector.
```

IBM Pascal

```
Vectors (.n.).V4(.1.):= vector n x-component
Vectors (.n.).V4(.2.):= vector n y-component
Vectors (.n.).V4(.3.):= vector n z-component
Vectors (.n.).V4(.4.):= vector n intensity (or hue)
   0 <= Vectors (.n.).V4(.4.) <=1 or
   0 <= Vectors (.n.).V4(.4.) <=127 if the vector list is tabulated.
Vectors (.n.).draw :=TRUE if vector n is a draw/line vector.
Vectors (.n.).draw :=FALSE if vector n is a move/position vector.
```

UNIX/C

```
Vectors [n].V4[0]:= vector n x-component
Vectors [n].V4[1]:= vector n y-component
Vectors [n].V4[2]:= vector n z-component
Vectors [n].V4[3]:= vector n intensity (or hue)
   0 <= Vectors [n].V4[3] <=1 or
   0 <= Vectors [n].V4[3] <=127 if the vector list is tabulated.
Vectors [n].draw :=TRUE if vector n is a draw/line vector.
Vectors [n].draw :=FALSE if vector n is a move/position vector.
```

The fourth position of **Vectors** is the intensity of that vector if vector-normalized, regardless of dimension. If block-normalized, the first vector's fourth position is used as the entire vector list intensity.

Together, the above 3 procedures implement the following PS 390 command.

PS 390 Command and Syntax

```
Name  := VECTOR_LIST [options][N=n]vectors;
```

VAX and IBM FORTRAN GSR

```
CALL PVcMax  (Max, ErrHnd)
```

where:

Max is a REAL*4

VAX PASCAL GSR

```
[GLOBAL, CHECK(NOBOUNDS)] PROCEDURE PVecMax (Maxcomp  :  REAL;
                          (PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PVecMax (Maxcomp  :  REAL;
         (PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PVecMax(MaxComp)
```

where:

float MaxComp;

DESCRIPTION

This routine must be called to set the maximum component of a vector list for multiple calls to PVcList (PVecList) with block-normalized vectors.

PS 390 Command and Syntax

```
Name  := VECTOR_LIST [options][N=n]vectors;
```

VAX and IBM FORTRAN GSR

```
CALL PViewP (Name, XMin, XMax, YMin, YMax, IMin, IMax, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
XMin, Xmax (horizontal) are REAL*4
YMin, Ymax (vertical) are REAL*4
IMin, IMax are REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PViewP (    %DESCR  Name       : P_VaryingType;
                              Xmin       : REAL;
                              Xmax       : REAL;
                              Ymin       : REAL;
                              Ymax       : REAL;
                              Imin       : REAL;
                              Imax       : REAL;
                      %DESCR  AppliedTo  : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PViewP (     CONST Name       : STRING;
                             Xmin       : SHORTREAL;
                             Xmax       : SHORTREAL;
                             Ymin       : SHORTREAL;
                             Ymax       : SHORTREAL;
                             Imin       : SHORTREAL;
                             Imax       : SHORTREAL;
                       CONST AppliedTo  : STRING;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PViewP(name,xmin,xmax,ymin,ymax,imin,imax,appliedto)
```

where:

```
string name,appliedto;
double  xmin,xmax,ymin,ymax,imin,imax;
```

DESCRIPTION

This routine specifies the area of the screen that the displayed data will occupy, and the range of intensity of the lines. It contains the following parametric definitions:

- Xmin, Xmax specify the horizontal boundaries of the new viewport
- Ymin, Ymax specify the vertical boundaries of the new viewport
- Imin, Imax specify the minimum and maximum intensities for the viewport.

PS 390 Command and Syntax

```
Name := VIEWport HORizontal = Xmin:Xmax
                 VERTical = Ymin:Ymax
                 [INTENsity] = Imin:Imax[APPLied to name1];
```

SEE ALSO

SET INTENSITY

VAX and IBM FORTRAN GSR

```
CALL PWindo (Name, Xmin, Xmax, Ymin, Ymax, Front, Back, Apply, ErrHnd)
```

where:

Name is a CHARACTER STRING
XMin, Xmax (horizontal) are REAL*4
YMin, Ymax (vertical) are REAL*4
Front is a REAL*4
Back is a REAL*4
Apply is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

VAX PASCAL GSR

```
PROCEDURE PWindow (  %DESCR  Name       : P_VaryingType;
                             Xmin       : REAL;
                             Xmax       : REAL;
                             Ymin       : REAL;
                             Ymax       : REAL;
                             Front      : REAL;
                             Back       : REAL;
                        %DESCR  AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PWindow (     CONST Name       : STRING;
                              Xmin       : SHORTREAL;
                              Xmax       : SHORTREAL;
                              Ymin       : SHORTREAL;
                              Ymax       : SHORTREAL;
                              Front      : SHORTREAL;
                              Back       : SHORTREAL;
                            CONST AppliedTo : STRING;
                      PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PWindow(name,xmin,xmax,ymin,ymax,front,back,appliedto)
```

where:

string name,appliedto;
double xmin,xmax,ymin,ymax,front,back;

DESCRIPTION

This routine specifies a right rectangular prism enclosing a portion of the data space to be displayed in parallel projection. It contains the following parametric definitions:

- Xmin, Xmax (horizontal) specify the window's boundaries along the x axis
- Ymin, Ymax (vertical) specify the window's boundaries on the y axis
- Front specifies the front boundary
- Back specifies the back boundary

PS 390 Command and Syntax

```
Name  := WINDOW   X = Xmin:Xmax
                  Y = Ymin:Ymax
                  [FRONT boundary = zmin BACK boundary = zmax]
                  [APPLied to name1];
```

SEE ALSO

EYEBACK, FIELD_OF_VIEW

VAX and IBM FORTRAN GSR

```
CALL PWrtBk  ( Name, Name1, Errhnd)
```

where:

Name1 is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PWrtBack ( %DESCR Name  : P_VaryingType;
                     %DESCR Name1 : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PWrtBack ( CONST Name  : P_VaryingType;
                     CONST Name1 : P_VaryingType;
                     PROCEDURE Error_Handler (Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PWrtBack(name,name1)
```

where:

string name,name1;

DESCRIPTION

This routine enables writeback in the data structure **Name1**. Writeback is triggered by sending a TRUE to the writeback operation node created with this routine.

PS 390 Command and Syntax

```
name := WRITEBACK  [APPLied to Name1];
```

VAX and IBM FORTRAN GSR

```
CALL PXfCan (Name, Apply, ErrHnd)
```

where

Name is a CHARACTER STRING
Apply is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PXfCancl (    %DESCR  Name      : P_VaryingType;
                        %DESCR  AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PXfCancl (    CONST Name       : STRING;
                        CONST AppliedTo  : STRING;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PXfCancl(name,appliedto)
```

where:

string name,appliedto;

DESCRIPTION

This routine stops transform data processing of subsequent nodes.

PS 390 Command and Syntax

```
Name := CANCEL XFORM [APPLIED TO name1];
```

SEE ALSO

XFORM VECTOR_LIST, XFORM MATRIX

VAX and IBM FORTRAN GSR

```
CALL PXfMat (Name, Apply, ErrHnd)
```

where

Name is a CHARACTER STRING
Apply is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PXfMatrx (    %DESCR  Name      : P_VaryingType;
                        %DESCR  AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PXfMatrx (    CONST Name       : STRING;
                        CONST AppliedTo  : STRING;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PXfMatrx(name,appliedto)
```

where:

string name,appliedto;

DESCRIPTION

This routine allows subsequent nodes to be processed to produce a transformation matrix.

PS 390 Command and Syntax

```
Name := XFORM output_data_type [APPLied TO name1];
```

SEE ALSO

XFORM VECTOR_LIST, XFORM CANCEL

VAX and IBM FORTRAN GSR

```
CALL PXfVec (Name, Apply, ErrHnd)
```

where

Name is a CHARACTER STRING
Apply is a CHARACTER STRING
Errhnd is the user-defined error-handler subroutine

VAX PASCAL GSR

```
PROCEDURE PXfVectr (    %DESCR  Name      : P_VaryingType;
                        %DESCR  AppliedTo : P_VaryingType;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

IBM PASCAL GSR

```
PROCEDURE PXfVectr (    CONST Name       : STRING;
                        CONST AppliedTo  : STRING;
                    PROCEDURE Error_Handler ( Err : INTEGER));
```

UNIX/C GSR

```
#include <ps300/gsrext.h>

PXfVectr(name,appliedto)
```

where:

string name,appliedto;

DESCRIPTION

This routine allows subsequent nodes to be processed to produce a transformed vector_list.

PS 390 Command and Syntax

```
Name := XFORM output_data_type [APPLied TO name1;
```

SEE ALSO

CANCEL XFORM, XFORM MATRIX

Appendix A
GSRs and Corresponding ASCII Commands

This appendix contains a list of the GSRs and the corresponding ASCII command name. The names of the GSRs and the corresponding utility or raster routine are also included. ASCII command descriptions will be found in Section *RM1*.

The user should note the following when using this appendix:

The left three columns list the FORTRAN, Pascal and UNIX/C GSRs in alphabetical order with the FORTRAN names. The right column lists the corresponding ASCII command name or utility or raster routine name. N/A means that there is no GSR.

In general, there is a one-to-one correspondence between GSRs and the corresponding ASCII command. The following three ASCII commands require more than one GSR:

LABELS
POLYGON
VECTOR_LIST

The utility and raster routines do not have a corresponding ASCII command.

ASCII commands with different parameters have separate GSRs. For example, the ROTATE command has the following three GSRs:

- PRotX (ROTATE IN X)
- PRotY (ROTATE IN Y)
- PRotZ (ROTATE IN Z)

FORTRAN	Pascal	UNIX/C	ASCII Command / Routine Name
PAttch	PAttach	PAttach	Attach PS 390 to Communication Device - utility GSR
PAttr	PAttrib	PAttrib	ATTRIBUTES
PAttr2	PAttrib2	PAttrib2	ATTRIBUTES
PBeg	PBegin	PBegin	BEGIN...END
PBegS	PBeginS	PBeginS	BEGIN_STRUCTURE...END_STRUCTURE
PBspl	PBspl	PBspl	BSPLINE
PChRot	PCharRot	PCharRot	CHARACTER ROTATE
PChs	PChars	PChars	CHARACTERS
PChSca	PCharSca	PCharSca	CHARACTER SCALE
PConn	PConnect	PConnect	CONNECT
PCopyV	PCopyVec	PCopyVec	COPY
PDefPa	PDefPatt	PDefPatt	PATTERN
PDelet	PDelete	PDelete	DELETE
PDelim	N/A	N/A	Set Delimiting Character - utility GSR
PDeLOD	PDecLOD	PDecLOD	DECREMENT LEVEL_OF_DETAIL
PDelW	PDelWild	PDelWild	DELETE
PDi	PDisc	PDisc	DISCONNECT
PDiAll	PDiscAll	PDiscAll	DISCONNECT
PDInfo	PDevInfo	N/A	Query GSR Device Status - utility GSR
PDiOut	PDiscOut	PDiscOut	DISCONNECT
PDisp	PDisplay	PDisplay	DISPLAY
PDtach	PDetach	PDetach	Detach PS 390 from Communication Device - utility GSR
PEnd	PEnd	PEnd	BEGIN...END
PEndOp	PEndOpt	PEndOpt	OPTIMIZE STRUCTURE;...END OPTIMIZE;

(continued)

FORTRAN	Pascal	UNIX/C	ASCII Command / Routine Name
PEndS	PEndS	PEndS	BEGIN_STRUCTURE...END_STRUCTURE
PEraPa	PEraPatt	PEraPatt	ERASE PATTERN FROM
PEyeBk	PEyeBack	PEyeBack	EYE BACK
PFn	PFnInst	PFnInst	(Function Instancing)
PFnN	PFnInstN	PFnInstN	(Function Instancing)
PFoll	PFoll	PFoll	FOLLOW WITH
PFont	PFont	PFont	CHARACTER FONT
PForg	PForget	PForget	FORGET (Structures)
PFov	PFov	PFov	FIELD_OF_VIEW
PGet	PGet	PGet	Poll PS 390 for Messages - utility GSR
PGetW	PGetWait	PGetWait	Read Messages from PS 390 - utility GSR
PGUCPU	PGiveUpCPU	PGiveUpCPU	GIVE_UP_CPU
PIfBit	PIfBit	PIfBit	IF CONDITIONAL_BIT
PIfLev	PIfLevel	PIfLevel	IF LEVEL_OF_DETAIL
PIfPha	PIfPhase	PIfPhase	IF PHASE
PIllum	PIllumin	PIllumin	ILLUMINATION
PIncl	PIncl	PIncl	INCLUDE
PInit	PInit	PInit	INITIALIZE
PInitC	PInitC	PInitC	INITIALIZE
PInitD	PInitD	PInitD	INITIALIZE
PInitN	PInitN	PInitN	INITIALIZE
PInLOD	PIncLOD	PIncLOD	INCREMENT LEVEL_OF_DETAIL
PInst	PInst	PInst	INSTANCE OF

(continued)

FORTRAN	Pascal	UNIX/C	ASCII Command / Routine Name
PLaAdd	PLabAdd	PLabAdd	LABELS
PLaBeg	PLabBegn	PLabBegn	
PLaEnd	PLabEnd	PLabEnd	
N/A	N/A	PLoad	Load Saved GSR Data - utility GSR
PLookA	PLookAt	PLookAt	LOOK
PMat22	PMat2x2	PMat2x2	MATRIX_2x2
PMat33	PMat3x3	PMat3x3	MATRIX_3x3
PMat43	PMat4x3	PMat4x3	MATRIX_4x3
PMat44	PMat4x4	PMat4x4	MATRIX_4x4
PMuxCI	PMuxCI	PMuxCI	Set Global Binary Output Channel-utility GSR
PMuxG	PMuxG	PMuxG	Set Global Generic Channel - utility GSR
PMuxP	PMuxPars	PMuxPars	Set Global Parser Channel - utility GSR
PNil	PNameNil	PNameNil	NIL
POpt	POptStru	POptStru	OPTIMIZE STRUCTURE;...END OPTIMIZE;
PPatWi	PPatWith	PPatWith	PATTERN WITH
PPlygA	PPlygAtr	PPlygAtr	POLYGON
PPlygB	PPlygBeg	PPlygBeg	
PPlygE	PPlygEnd	PPlygEnd	
PPlygH	PPlygHSI	PPlygLisHSI	
PPlygL	PPlygLis	PPlygLis	
PPlygO	PPlygOtl	PPlygOtl	
PPlygR	PPlygRGB	PPlygLisRGB	
PPoly	PPoly	PPoly	POLYNOMIAL
PPref	PPref	PPref	PREFIX WITH
PPurge	PPurge	PPurge	Purge Output Buffer - utility GSR
PPutG	PPutG	PPutG	Send Bytes to Generic Output Channel - utility GSR

(continued)

FORTRAN	Pascal	UNIX/C	ASCII Command / Routine Name
PPutGX	PPutGX	N/A	Send Bytes to Generic Output Channel - utility GSR
PPutP	PPutPars	PPutPars	Send Bytes to Parser Output Channel - utility GSR
PRasCp	PRasCp	PRasCp	Set Current Pixel Location - raster GSR
PRasEr	PRasEr	PRasEr	Erase Screen - raster GSR
PRasLd	PRasLd	PRasLd	Set Logical Device Coordinates - raster GSR
PRasWP	PRasWP	PRasWP	Load Pixel Data - raster GSR
PRawBl	PRawBloc	PRawBloc	RAWBLOCK
PRaWRP	PRaWRP	PRaWRP	Set Raster Mode to Write Pixel Data - raster GSR
PRBspl	PRBspl	PRBspl	RATIONAL BSPLINE
PRem	PRem	PRem	REMOVE
PRemFo	PRemFoll	PRemFoll	REMOVE FOLLOWER
PRemFr	PRemFrom	PRemFrom	REMOVE FROM
PRemPr	PRemPref	PRemPref	REMOVE PREFIX
PRotX	PRotX	PRotX	ROTATE
PRotY	PRotY	PRotY	ROTATE
PRotZ	PRotZ	PRotZ	ROTATE
PRPoly	PRPoly	PRPoly	RATIONAL POLYNOMIAL
PRsvSt	PRsvStor	PRsvStor	RESERVE_WORKING_STORAGE
N/A	N/A	PSavBeg	Begin Saving GSR Data - utility GSR
N/A	N/A	PSavEnd	End Saving GSR Data - utility GSR
PScale	PScaleBy	PScaleBy	SCALE
PSeBit	PSetBit	PSetBit	SET CONDITIONAL_BIT